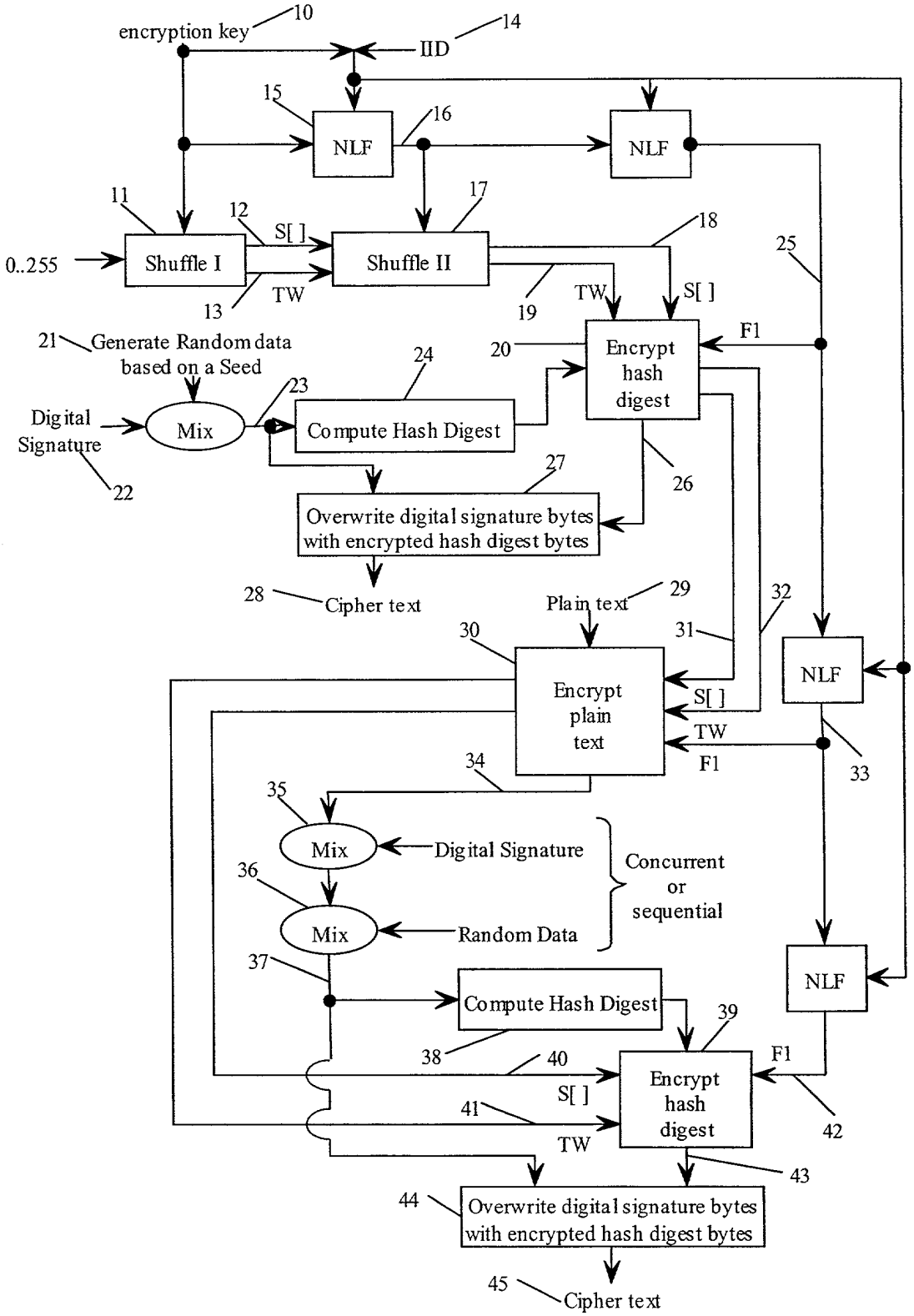

encryption key
10
IID
14
15
NLF
16
NLF
11
12
S[ ]
17
18
25
0..255
Shuffle I
Shuffle II
13
TW
19
TW
S[ ]
21
Generate Random data based on a Seed
20
Encrypt hash digest
F1
24
23
Digital Signature
Mix
Compute Hash Digest
22
27
26
Overwrite digital signature bytes with encrypted hash digest bytes
28
Cipher text
Plain text
29
32
30
31
NLF
Encrypt plain text
S[ ]
TW
F1
33
34
35
Mix
Digital Signature
Concurrent or sequential
36
Mix
Random Data
37
NLF
Compute Hash Digest
39
38
40
F1
S[ ]
Encrypt hash digest
41
42
TW
43
44
Overwrite digital signature bytes with encrypted hash digest bytes
45
Cipher text

Figure 1A

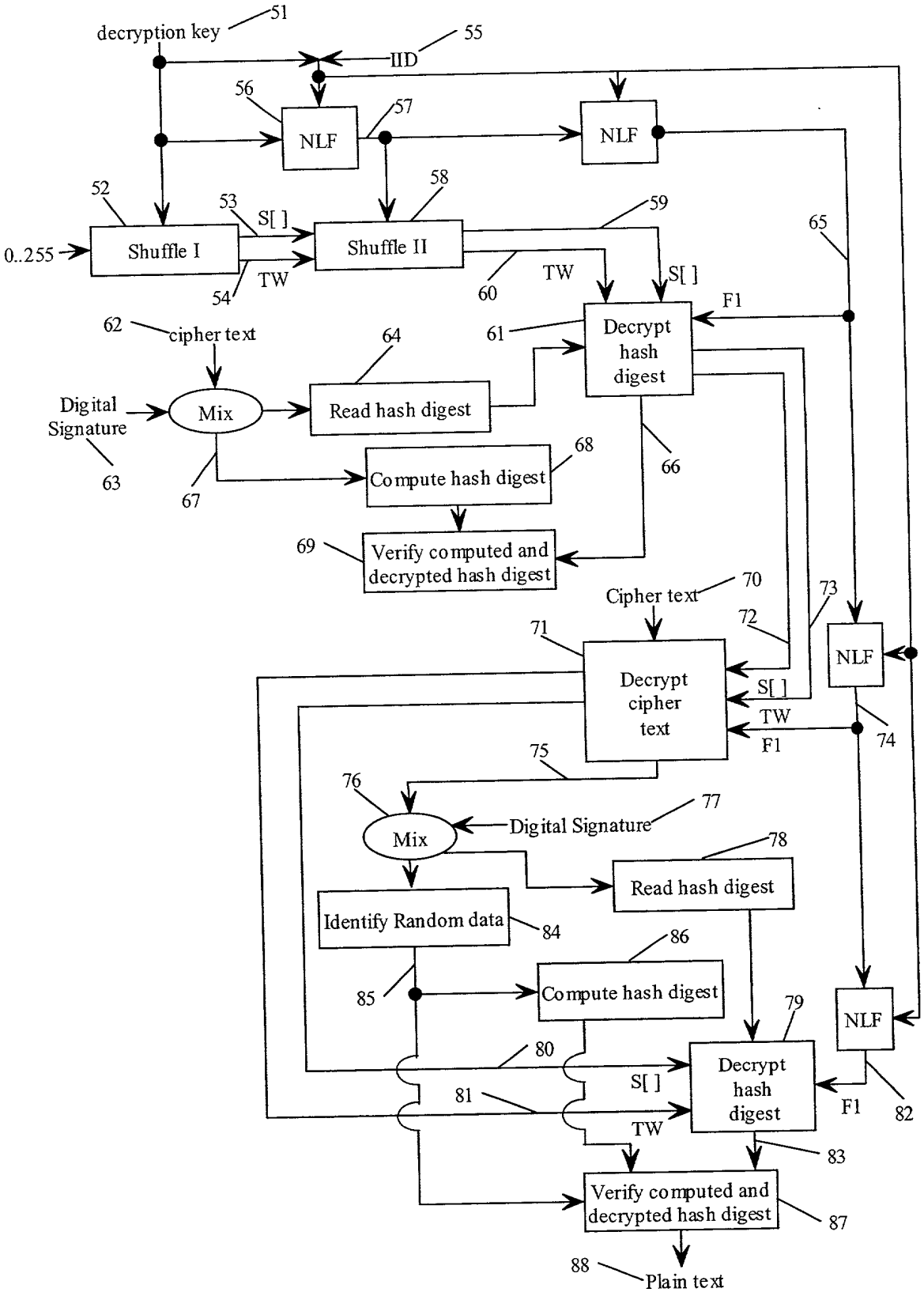

decryption key
51
IID
55
56
NLF
57
NLF
52
53
S[ ]
58
59
65
0..255
Shuffle I
Shuffle II
54
TW
60
TW
S[ ]
F1
62
cipher text
61
Decrypt hash digest
64
Digital Signature
Mix
Read hash digest
63
67
68
66
Compute hash digest
69
Verify computed and decrypted hash digest
Cipher text
70
73
72
71
NLF
Decrypt cipher text
S[ ]
TW
F1
74
75
76
77
Mix
Digital Signature
78
Read hash digest
Identify Random data
84
86
85
Compute hash digest
79
NLF
80
S[ ]
Decrypt hash digest
81
TW
F1
82
83
Verify computed and decrypted hash digest
87
88
Plain text

Figure 1B

Figure 2A

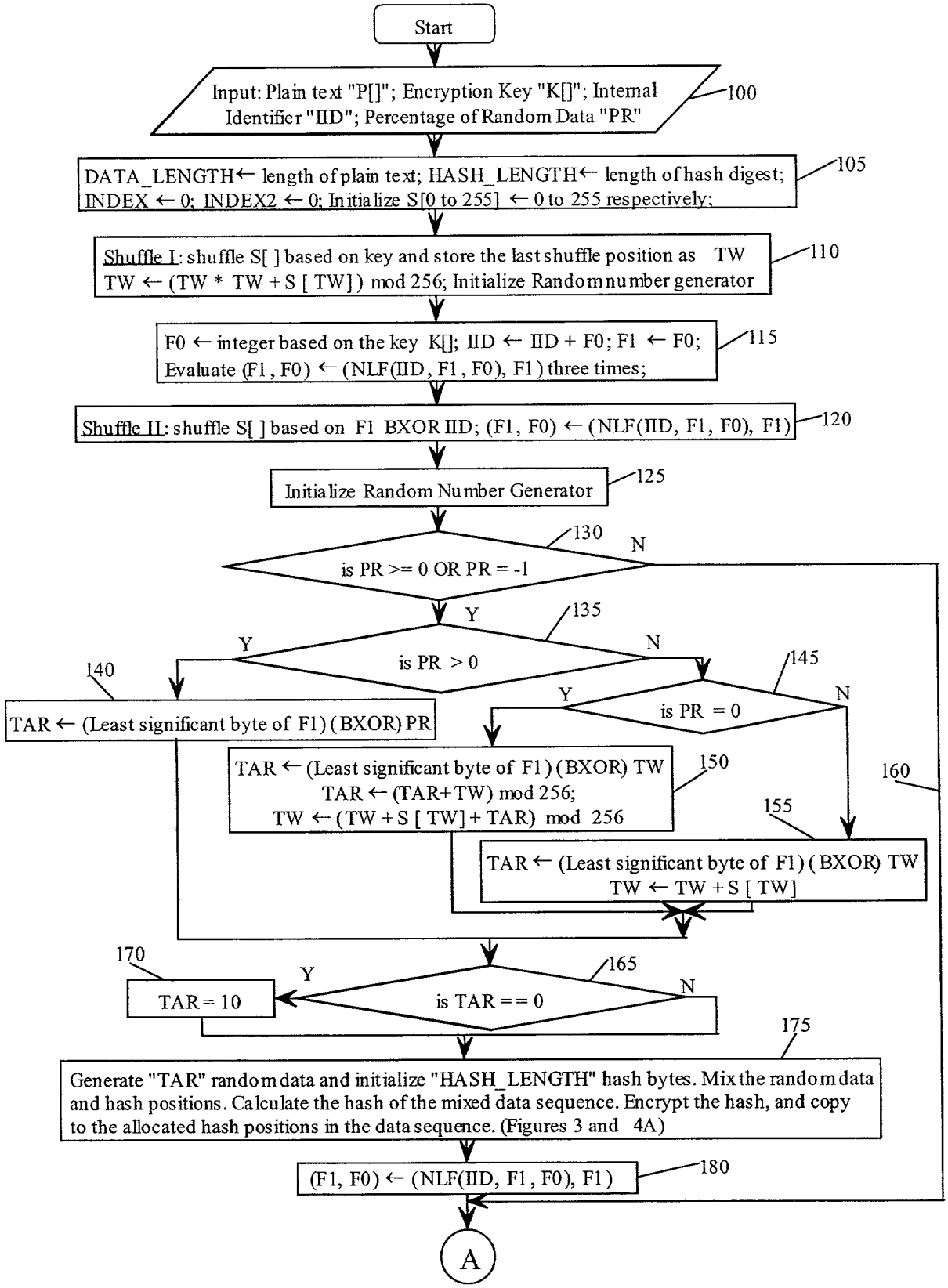
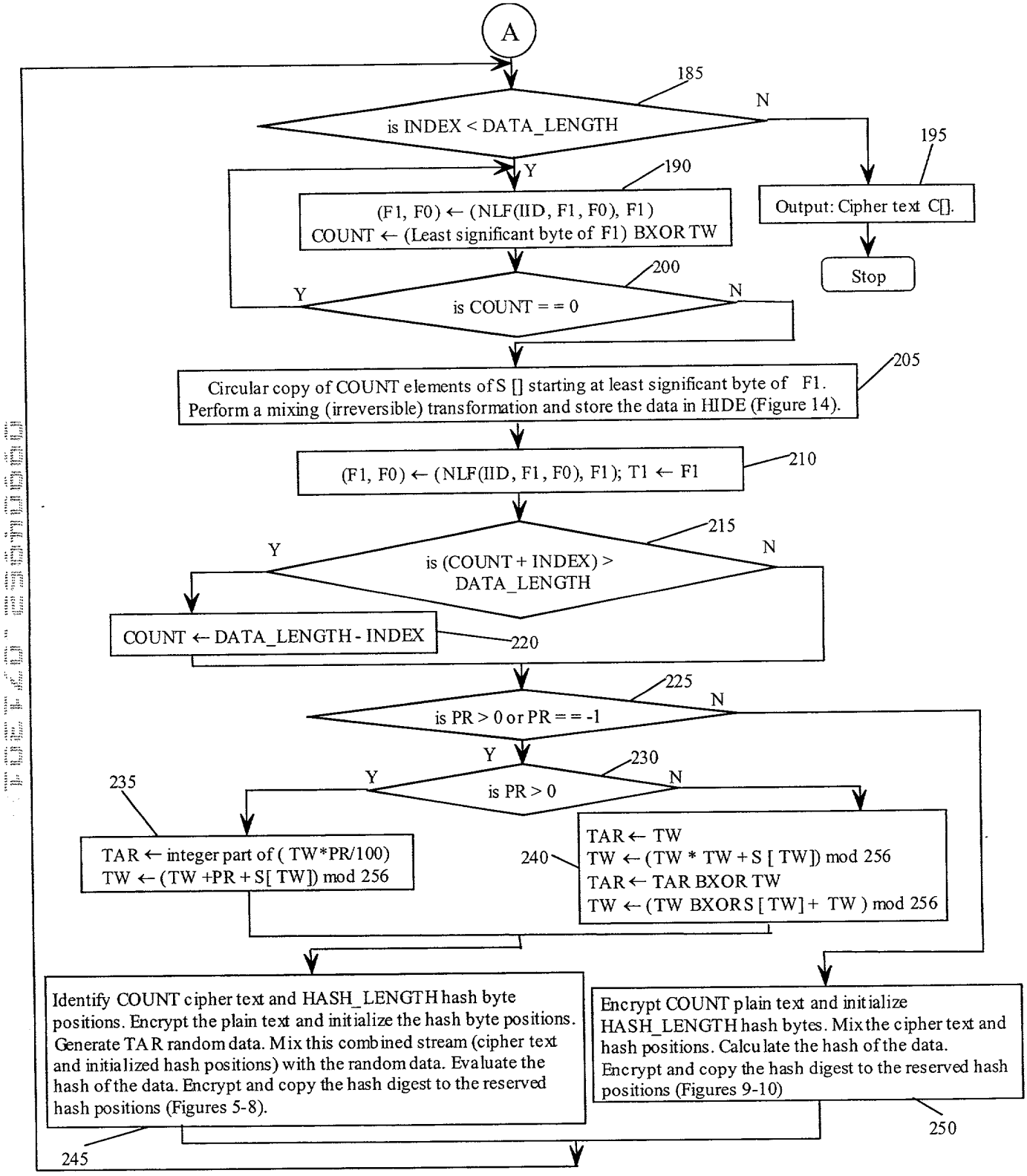

Start
Input: Plain text "P[]"; Encryption Key "K[]"; Internal Identifier "IID"; Percentage of Random Data "PR"
100
DATA_LENGTH← length of plain text; HASH_LENGTH← length of hash digest; INDEX ← 0; INDEX2 ← 0; Initialize S[0 to 255] ← 0 to 255 respectively;
105
Shuffle I: shuffle S[ ] based on key and store the last shuffle position as TW
TW ← (TW * TW + S [TW]) mod 256; Initialize Random number generator
110
F0 ← integer based on the key K[]; IID ← IID + F0; F1 ← F0;
Evaluate (F1, F0) ← (NLF(IID, F1, F0), F1) three times;
115
Shuffle II: shuffle S[ ] based on F1 BXOR IID; (F1, F0) ← (NLF(IID, F1, F0), F1)
120
Initialize Random Number Generator
125
130
is PR >= 0 OR PR = -1
N
Y
135
is PR > 0
Y
N
140
TAR ← (Least significant byte of F1) (BXOR) PR
145
is PR = 0
Y
N
TAR ← (Least significant byte of F1) (BXOR) TW
TAR ← (TAR+TW) mod 256;
TW ← (TW + S [TW] + TAR) mod 256
150
160
155
TAR ← (Least significant byte of F1) (BXOR) TW
TW ← TW + S [TW]
170
TAR = 10
Y
165
is TAR = = 0
N
175
Generate "TAR" random data and initialize "HASH_LENGTH" hash bytes. Mix the random data and hash positions. Calculate the hash of the mixed data sequence. Encrypt the hash, and copy to the allocated hash positions in the data sequence. (Figures 3 and 4A)
(F1, F0) ← (NLF(IID, F1, F0), F1)
180
A

Figure 2B


A
185
is INDEX < DATA_LENGTH
N
Y
195
Output: Cipher text C[].
Stop
190
(F1, F0) ← (NLF(IID, F1, F0), F1)
COUNT ← (Least significant byte of F1) BXOR TW
200
Y
is COUNT = = 0
N
205
Circular copy of COUNT elements of S [] starting at least significant byte of F1.
Perform a mixing (irreversible) transformation and store the data in HIDE (Figure 14).
210
(F1, F0) ← (NLF(IID, F1, F0), F1); T1 ← F1
215
Y
is (COUNT + INDEX) > DATA_LENGTH
N
COUNT ← DATA_LENGTH - INDEX
220
225
is PR > 0 or PR = = -1
N
Y
230
Y
is PR > 0
N
235
TAR ← integer part of (TW*PR/100)
TW ← (TW +PR + S[TW]) mod 256
240
TAR ← TW
TW ← (TW * TW + S [TW]) mod 256
TAR ← TAR BXOR TW
TW ← (TW BXORS [TW] + TW) mod 256
Identify COUNT cipher text and HASH_LENGTH hash byte positions. Encrypt the plain text and initialize the hash byte positions. Generate TAR random data. Mix this combined stream (cipher text and initialized hash positions) with the random data. Evaluate the hash of the data. Encrypt and copy the hash digest to the reserved hash positions (Figures 5-8).
Encrypt COUNT plain text and initialize HASH_LENGTH hash bytes. Mix the cipher text and hash positions. Calculate the hash of the data. Encrypt and copy the hash digest to the reserved hash positions (Figures 9-10)
250
245

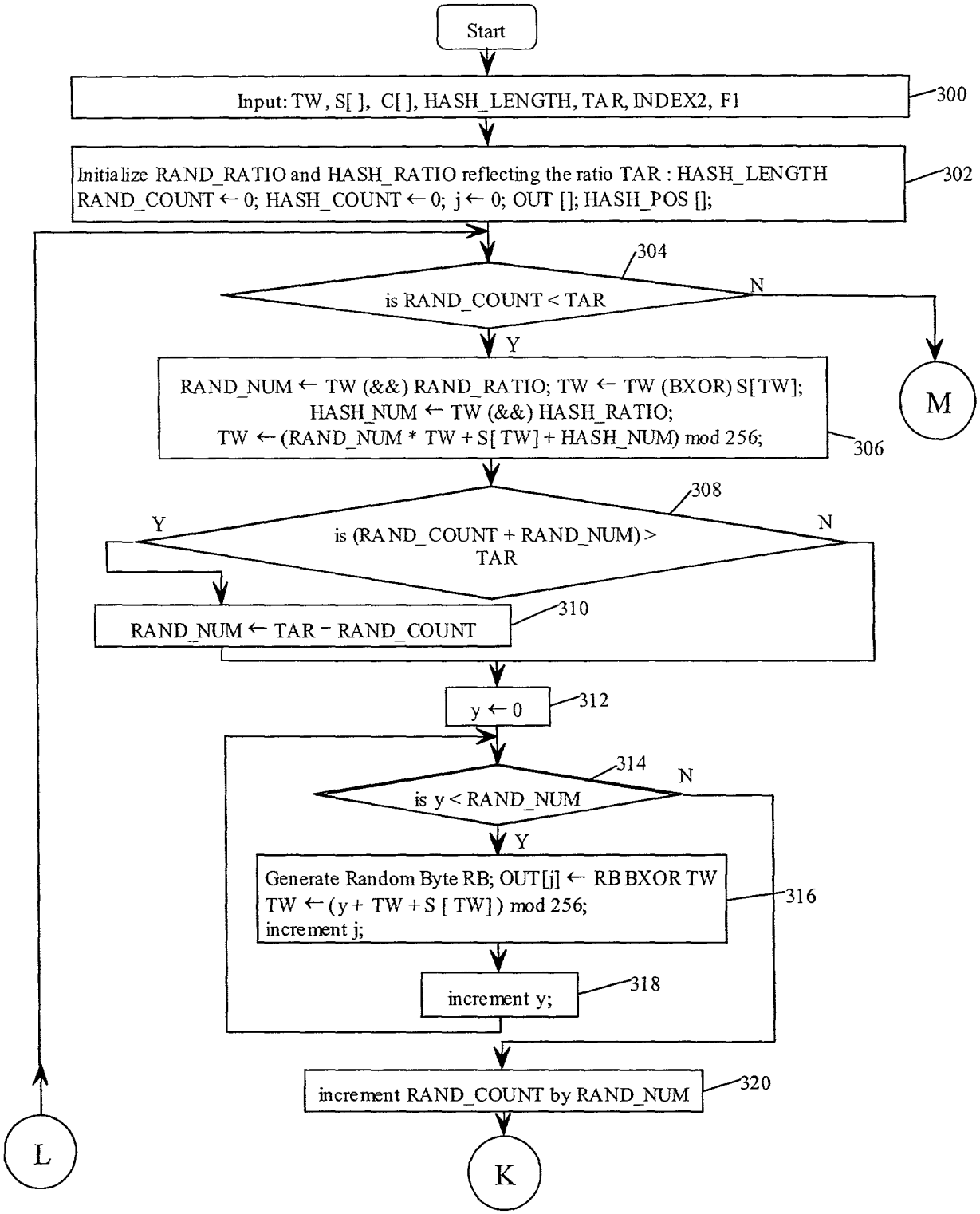

Start
Input: TW, S[ ], C[ ], HASH_LENGTH, TAR, INDEX2, F1
300
Initialize RAND_RATIO and HASH_RATIO reflecting the ratio TAR : HASH_LENGTH
RAND_COUNT ← 0; HASH_COUNT ← 0; j ← 0; OUT []; HASH_POS [];
302
304
is RAND_COUNT < TAR
N
Y
M
RAND_NUM ← TW (&&) RAND_RATIO; TW ← TW (BXOR) S[TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (RAND_NUM * TW + S[TW] + HASH_NUM) mod 256;
306
308
Y
is (RAND_COUNT + RAND_NUM) > TAR
N
RAND_NUM ← TAR - RAND_COUNT
310
y ← 0
312
314
is y < RAND_NUM
N
Y
Generate Random Byte RB; OUT[j] ← RB BXOR TW
TW ← (y + TW + S [TW]) mod 256;
increment j;
316
increment y;
318
increment RAND_COUNT by RAND_NUM
320
L
K

Figure 3

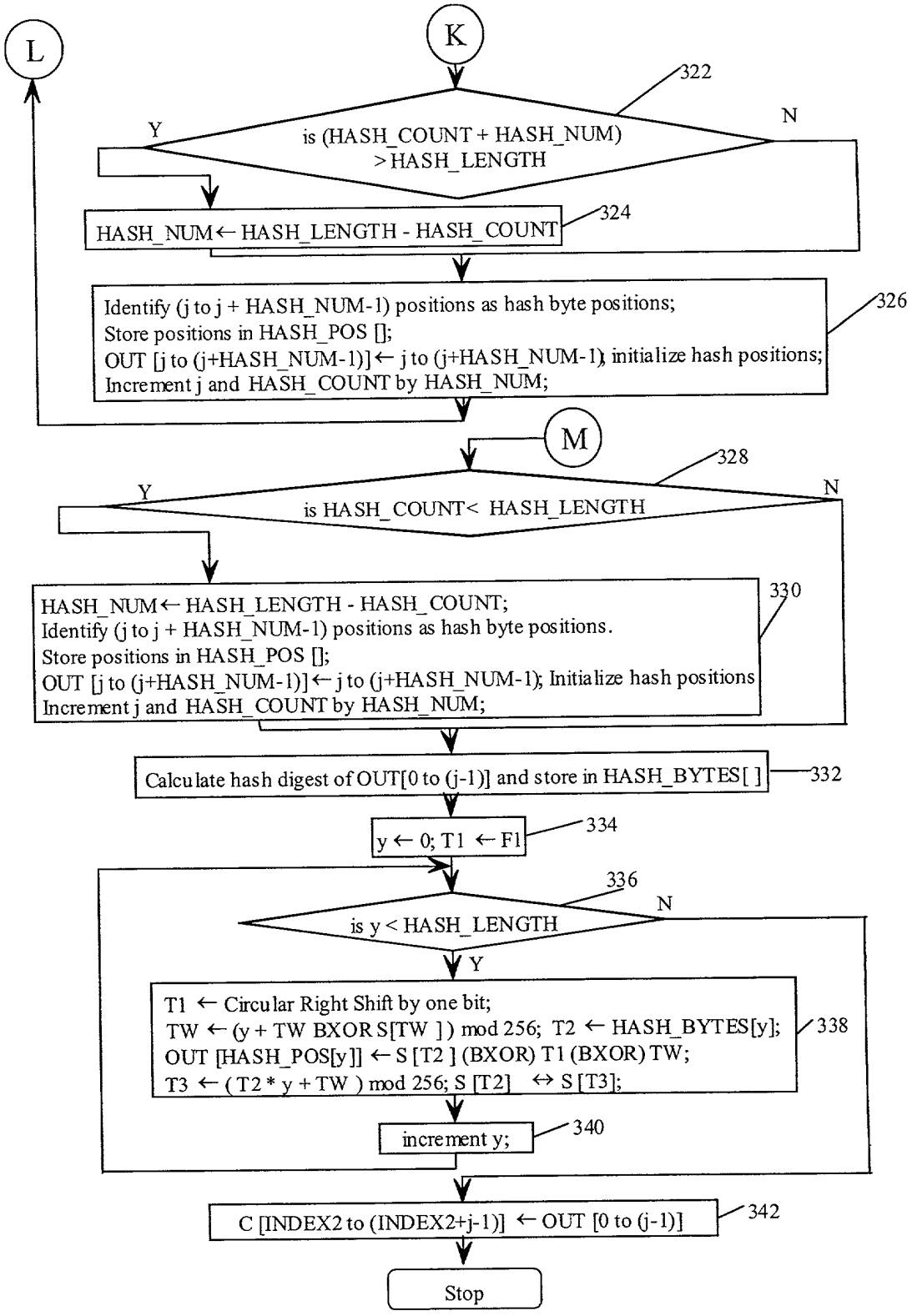

L
K
322
Y
is (HASH_COUNT + HASH_NUM) > HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
324
Identify (j to j + HASH_NUM-1) positions as hash byte positions;
Store positions in HASH_POS [];
OUT [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1); initialize hash positions;
Increment j and HASH_COUNT by HASH_NUM;
326
M
328
Y
is HASH_COUNT< HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT;
Identify (j to j + HASH_NUM-1) positions as hash byte positions.
Store positions in HASH_POS [];
OUT [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1); Initialize hash positions
Increment j and HASH_COUNT by HASH_NUM;
330
Calculate hash digest of OUT[0 to (j-1)] and store in HASH_BYTES[ ]
332
y ← 0; T1 ← F1
334
336
is y < HASH_LENGTH
N
Y
T1 ← Circular Right Shift by one bit;
TW ← (y + TW BXOR S[TW]) mod 256; T2 ← HASH_BYTES[y];
OUT [HASH_POS[y]] ← S [T2] (BXOR) T1 (BXOR) TW;
T3 ← (T2 * y + TW) mod 256; S [T2] ↔ S [T3];
338
increment y;
340
C [INDEX2 to (INDEX2+j-1)] ← OUT [0 to (j-1)]
342
Stop

Figure 4A

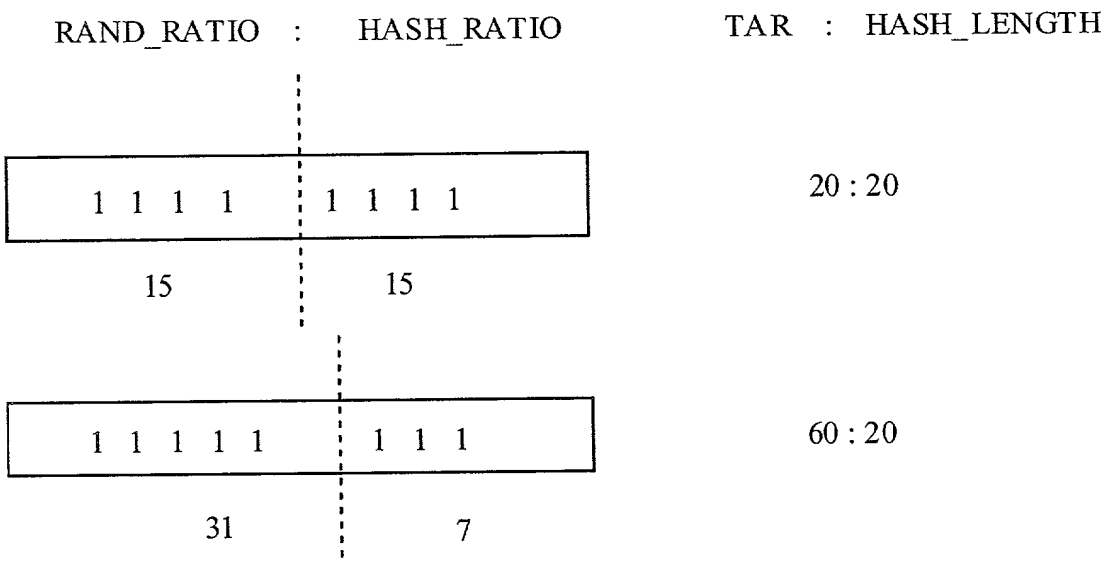

RAND_RATIO : HASH_RATIO
TAR : HASH_LENGTH
1 1 1 1
1 1 1 1
20 : 20
15
15
1 1 1 1 1
1 1 1
60 : 20
31
7

Figure 4B

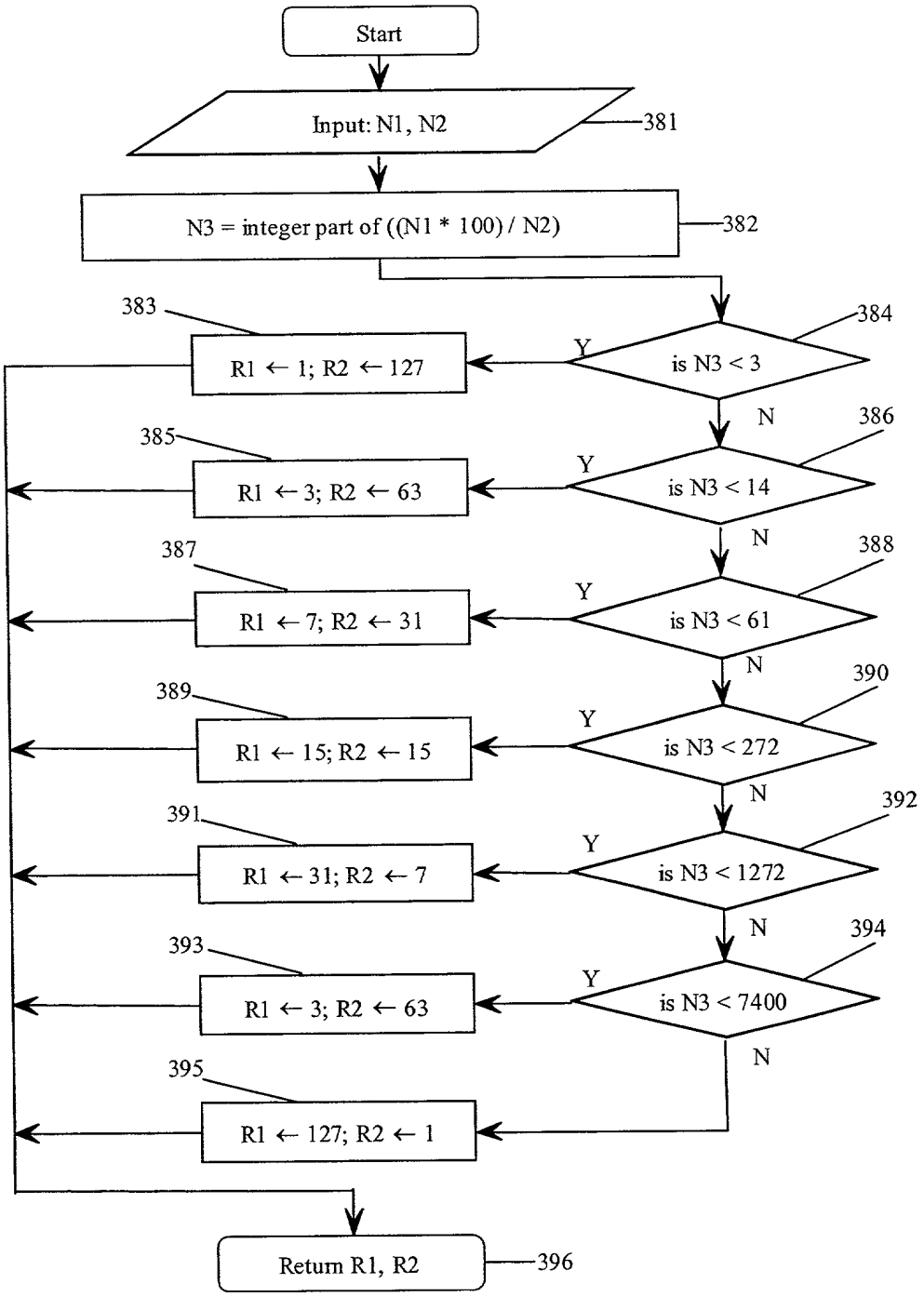

Start
Input: N1, N2
381
N3 = integer part of ((N1 * 100) / N2)
382
383
384
is N3 < 3
Y
R1 ← 1; R2 ← 127
N
385
386
is N3 < 14
Y
R1 ← 3; R2 ← 63
N
387
388
is N3 < 61
Y
R1 ← 7; R2 ← 31
N
389
390
is N3 < 272
Y
R1 ← 15; R2 ← 15
N
391
392
is N3 < 1272
Y
R1 ← 31; R2 ← 7
N
393
394
is N3 < 7400
Y
R1 ← 3; R2 ← 63
N
395
R1 ← 127; R2 ← 1
Return R1, R2
396

Figure 4C

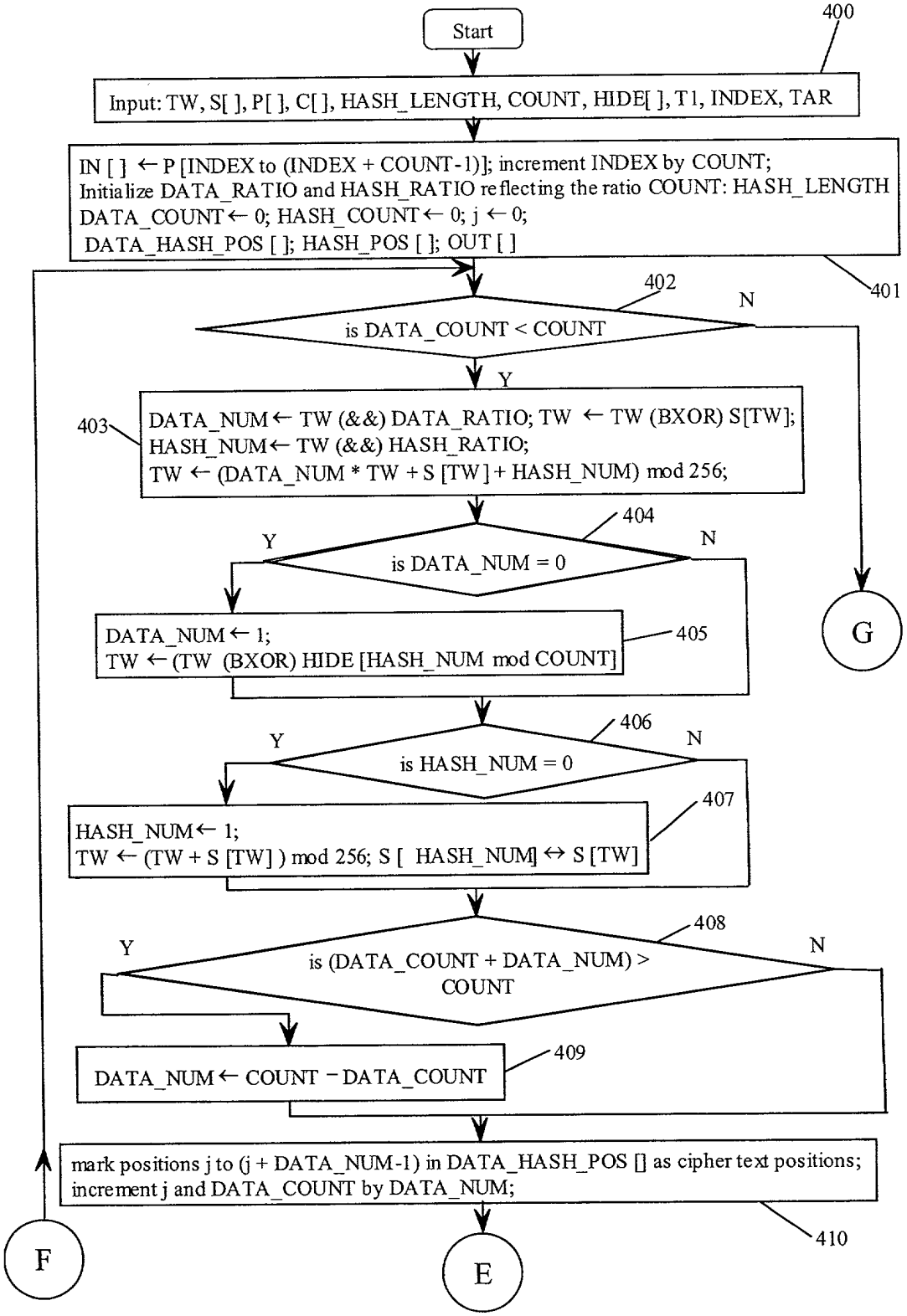

Start
400
Input: TW, S[ ], P[ ], C[ ], HASH_LENGTH, COUNT, HIDE[ ], T1, INDEX, TAR
IN [ ] ← P [INDEX to (INDEX + COUNT-1)]; increment INDEX by COUNT;
Initialize DATA_RATIO and HASH_RATIO reflecting the ratio COUNT: HASH_LENGTH
DATA_COUNT ← 0; HASH_COUNT ← 0; j ← 0;
DATA_HASH_POS [ ]; HASH_POS [ ]; OUT [ ]
401
402
is DATA_COUNT < COUNT
N
Y
403
DATA_NUM ← TW (&&) DATA_RATIO; TW ← TW (BXOR) S[TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (DATA_NUM * TW + S [TW] + HASH_NUM) mod 256;
404
Y
is DATA_NUM = 0
N
G
DATA_NUM ← 1;
TW ← (TW (BXOR) HIDE [HASH_NUM mod COUNT]
405
406
Y
is HASH_NUM = 0
N
HASH_NUM ← 1;
TW ← (TW + S [TW]) mod 256; S [HASH_NUM] ↔ S [TW]
407
408
Y
is (DATA_COUNT + DATA_NUM) > COUNT
N
DATA_NUM ← COUNT − DATA_COUNT
409
mark positions j to (j + DATA_NUM-1) in DATA_HASH_POS [] as cipher text positions;
increment j and DATA_COUNT by DATA_NUM;
410
F
E

Figure 5

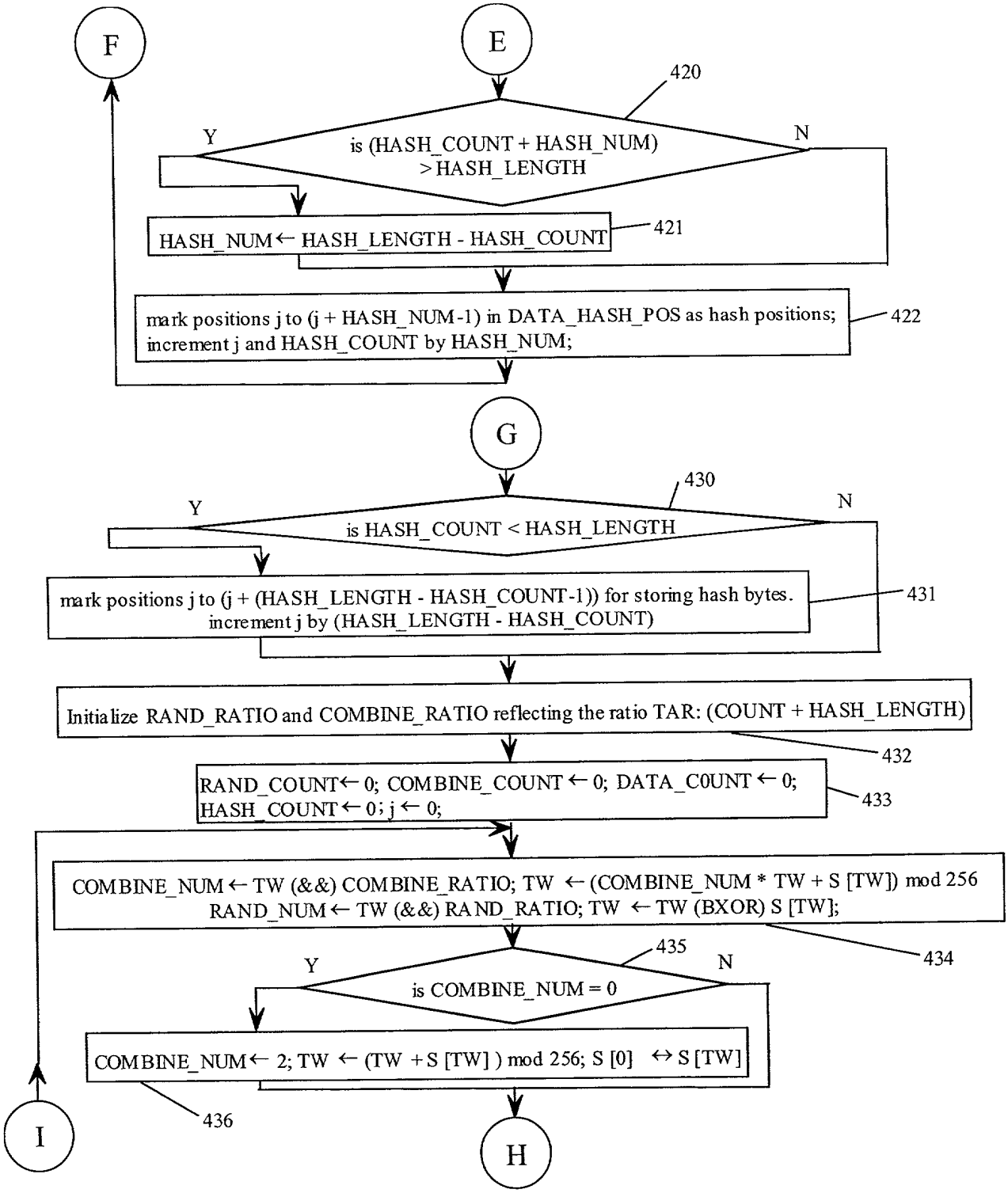

F
E
420
Y
is (HASH_COUNT + HASH_NUM) > HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
421
mark positions j to (j + HASH_NUM-1) in DATA_HASH_POS as hash positions; increment j and HASH_COUNT by HASH_NUM;
422
G
430
Y
is HASH_COUNT < HASH_LENGTH
N
mark positions j to (j + (HASH_LENGTH - HASH_COUNT-1)) for storing hash bytes. increment j by (HASH_LENGTH - HASH_COUNT)
431
Initialize RAND_RATIO and COMBINE_RATIO reflecting the ratio TAR: (COUNT + HASH_LENGTH)
432
RAND_COUNT ← 0; COMBINE_COUNT ← 0; DATA_COUNT ← 0; HASH_COUNT ← 0; j ← 0;
433
COMBINE_NUM ← TW (&&) COMBINE_RATIO; TW ← (COMBINE_NUM * TW + S [TW]) mod 256
RAND_NUM ← TW (&&) RAND_RATIO; TW ← TW (BXOR) S [TW];
434
435
Y
is COMBINE_NUM = 0
N
COMBINE_NUM ← 2; TW ← (TW + S [TW]) mod 256; S [0] ↔ S [TW]
436
I
H

Figure 6

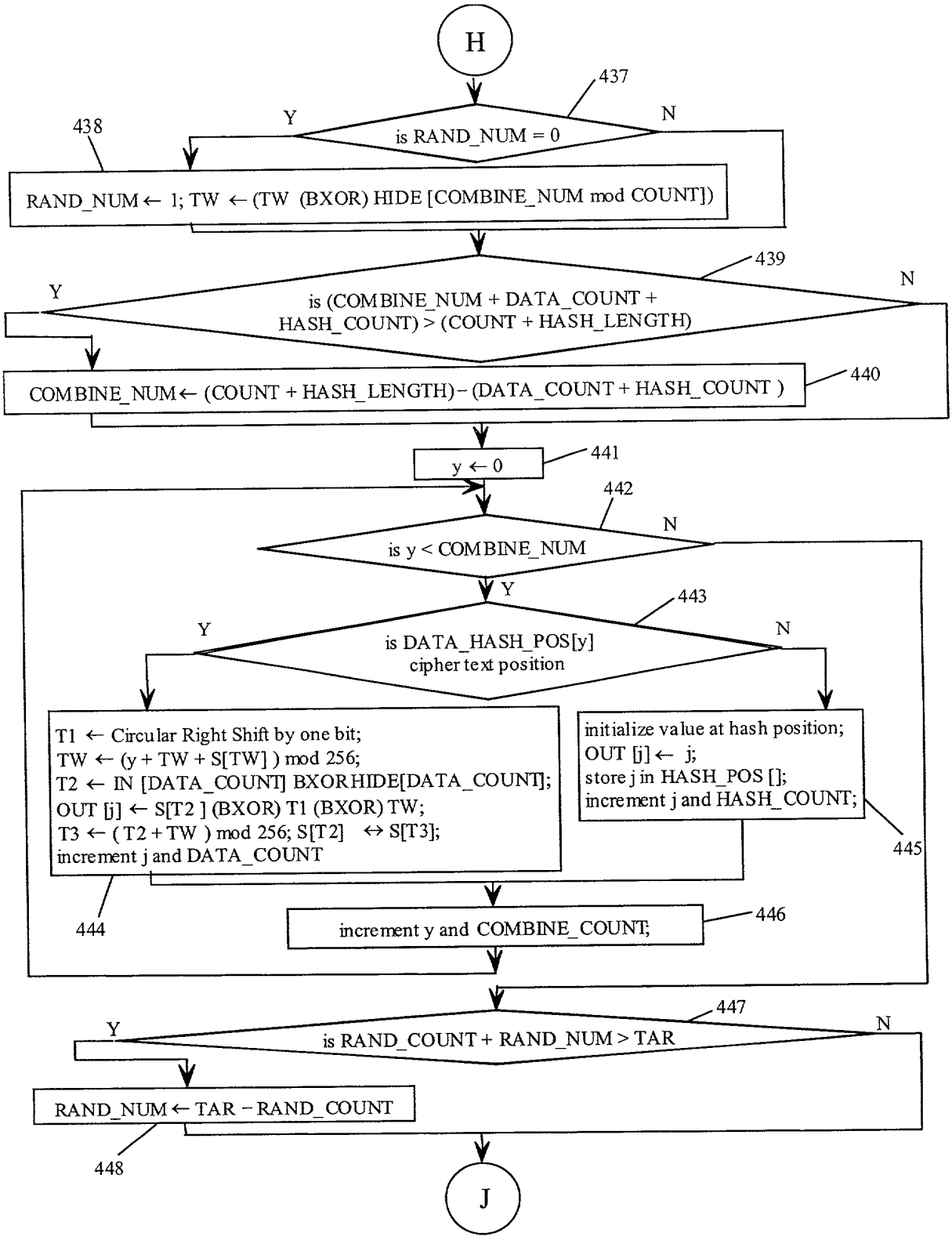

H
437
is RAND_NUM = 0
Y
N
438
RAND_NUM ← 1; TW ← (TW (BXOR) HIDE [COMBINE_NUM mod COUNT])
439
is (COMBINE_NUM + DATA_COUNT + HASH_COUNT) > (COUNT + HASH_LENGTH)
Y
N
COMBINE_NUM ← (COUNT + HASH_LENGTH) − (DATA_COUNT + HASH_COUNT)
440
y ← 0
441
442
is y < COMBINE_NUM
N
Y
443
is DATA_HASH_POS[y] cipher text position
Y
N
T1 ← Circular Right Shift by one bit;
TW ← (y + TW + S[TW]) mod 256;
T2 ← IN [DATA_COUNT] BXORHIDE[DATA_COUNT];
OUT [j] ← S[T2] (BXOR) T1 (BXOR) TW;
T3 ← (T2 + TW) mod 256; S[T2] ↔ S[T3];
increment j and DATA_COUNT
444
initialize value at hash position;
OUT [j] ← j;
store j in HASH_POS [];
increment j and HASH_COUNT;
445
increment y and COMBINE_COUNT;
446
447
is RAND_COUNT + RAND_NUM > TAR
Y
N
RAND_NUM ← TAR − RAND_COUNT
448
J

Figure 7

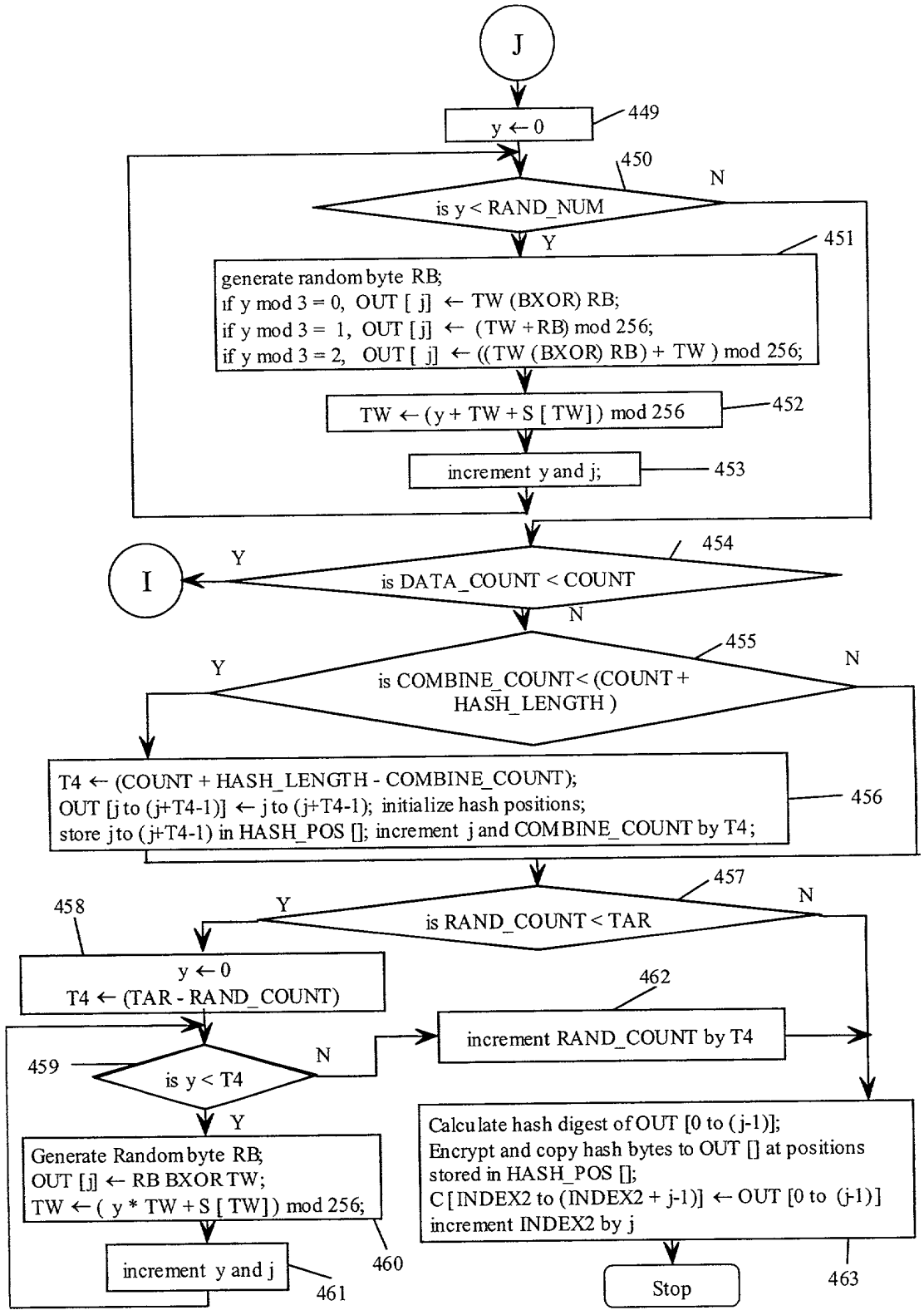

J
y ← 0
449
450
N
is y < RAND_NUM
Y
451
generate random byte RB;
if y mod 3 = 0, OUT [j] ← TW (BXOR) RB;
if y mod 3 = 1, OUT [j] ← (TW + RB) mod 256;
if y mod 3 = 2, OUT [j] ← ((TW (BXOR) RB) + TW) mod 256;
TW ← (y + TW + S [TW]) mod 256
452
increment y and j;
453
I
Y
454
is DATA_COUNT < COUNT
N
455
Y
N
is COMBINE_COUNT< (COUNT + HASH_LENGTH)
T4 ← (COUNT + HASH_LENGTH - COMBINE_COUNT);
OUT [j to (j+T4-1)] ← j to (j+T4-1); initialize hash positions;
store j to (j+T4-1) in HASH_POS []; increment j and COMBINE_COUNT by T4;
456
457
458
Y
N
is RAND_COUNT < TAR
y ← 0
T4 ← (TAR - RAND_COUNT)
462
increment RAND_COUNT by T4
459
N
is y < T4
Y
Generate Random byte RB;
OUT [j] ← RB BXOR TW;
TW ← (y * TW + S [TW]) mod 256;
460
increment y and j
461
Calculate hash digest of OUT [0 to (j-1)];
Encrypt and copy hash bytes to OUT [] at positions stored in HASH_POS [];
C [INDEX2 to (INDEX2 + j-1)] ← OUT [0 to (j-1)]
increment INDEX2 by j
463
Stop

Figure 8

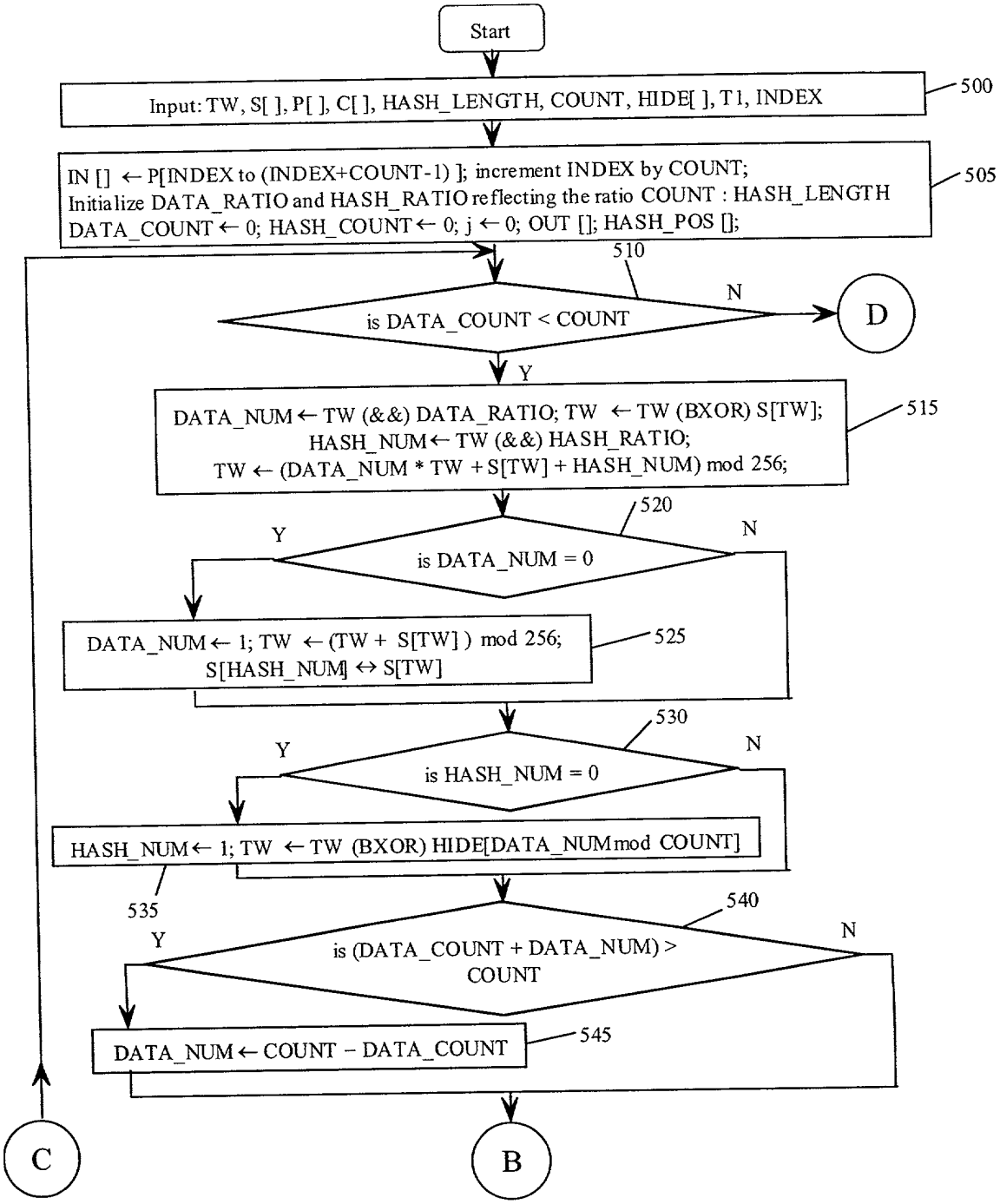

Start
Input: TW, S[ ], P[ ], C[ ], HASH_LENGTH, COUNT, HIDE[ ], T1, INDEX
500
IN [] ← P[INDEX to (INDEX+COUNT-1)]; increment INDEX by COUNT;
Initialize DATA_RATIO and HASH_RATIO reflecting the ratio COUNT : HASH_LENGTH
DATA_COUNT ← 0; HASH_COUNT ← 0; j ← 0; OUT []; HASH_POS [];
505
510
is DATA_COUNT < COUNT
N
D
Y
DATA_NUM ← TW (&&) DATA_RATIO; TW ← TW (BXOR) S[TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (DATA_NUM * TW + S[TW] + HASH_NUM) mod 256;
515
520
Y
is DATA_NUM = 0
N
DATA_NUM ← 1; TW ← (TW + S[TW]) mod 256;
S[HASH_NUM] ↔ S[TW]
525
530
Y
is HASH_NUM = 0
N
HASH_NUM ← 1; TW ← TW (BXOR) HIDE[DATA_NUM mod COUNT]
535
540
Y
is (DATA_COUNT + DATA_NUM) >
COUNT
N
DATA_NUM ← COUNT − DATA_COUNT
545
C
B

Figure 9

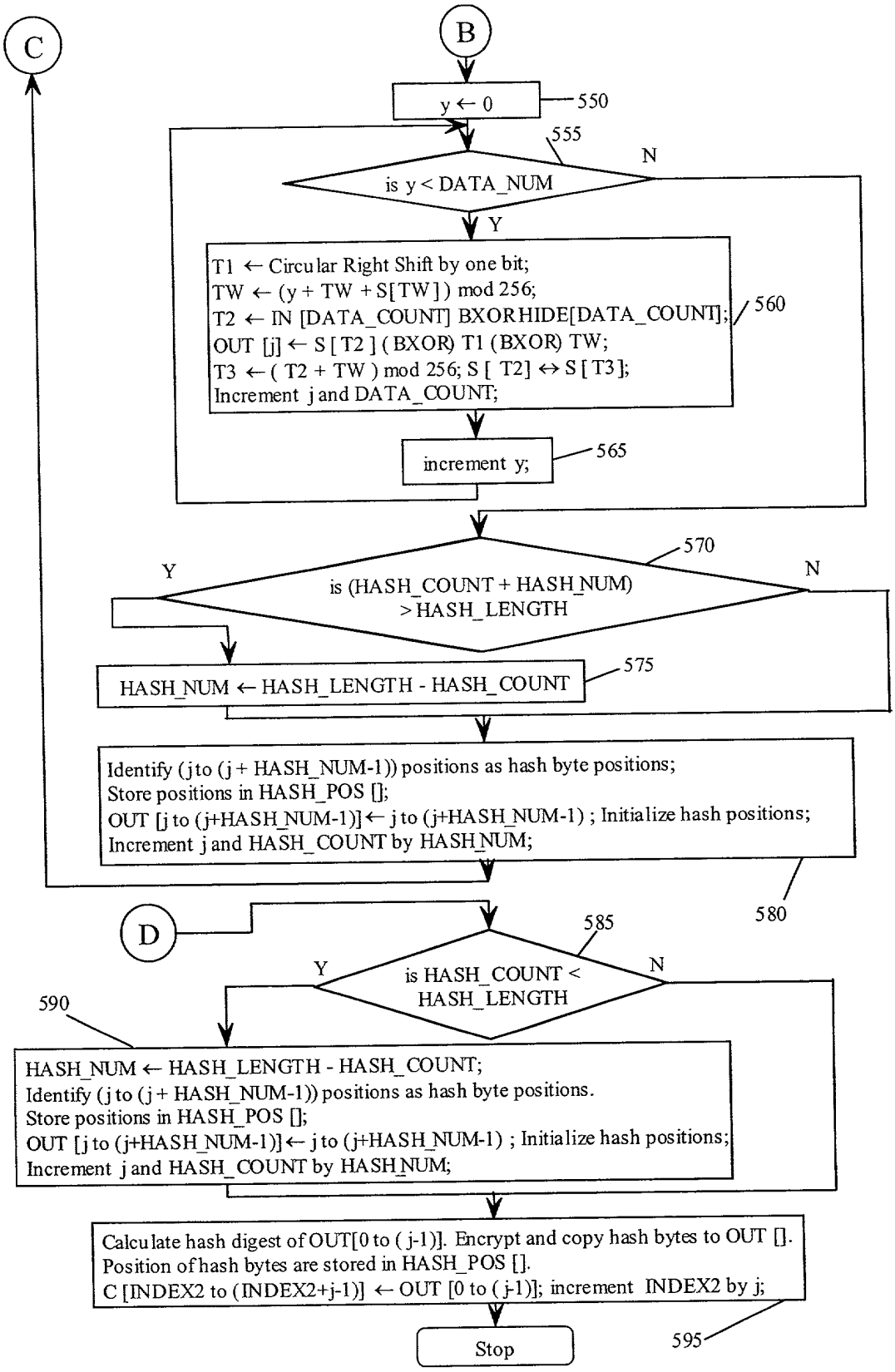

C
B
y ← 0
550
555
is y < DATA_NUM
N
Y
T1 ← Circular Right Shift by one bit;
TW ← (y + TW + S[TW]) mod 256;
T2 ← IN [DATA_COUNT] BXORHIDE[DATA_COUNT];
OUT [j] ← S [T2] (BXOR) T1 (BXOR) TW;
T3 ← (T2 + TW) mod 256; S [T2] ↔ S [T3];
Increment j and DATA_COUNT;
560
increment y;
565
570
Y
is (HASH_COUNT + HASH_NUM)
> HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
575
Identify (j to (j + HASH_NUM-1)) positions as hash byte positions;
Store positions in HASH_POS [];
OUT [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1) ; Initialize hash positions;
Increment j and HASH_COUNT by HASH_NUM;
580
D
585
Y
is HASH_COUNT <
HASH_LENGTH
N
590
HASH_NUM ← HASH_LENGTH - HASH_COUNT;
Identify (j to (j + HASH_NUM-1)) positions as hash byte positions.
Store positions in HASH_POS [];
OUT [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1) ; Initialize hash positions;
Increment j and HASH_COUNT by HASH_NUM;
Calculate hash digest of OUT[0 to (j-1)]. Encrypt and copy hash bytes to OUT [].
Position of hash bytes are stored in HASH_POS [].
C [INDEX2 to (INDEX2+j-1)] ← OUT [0 to (j-1)]; increment INDEX2 by j;
Stop
595

Figure 10

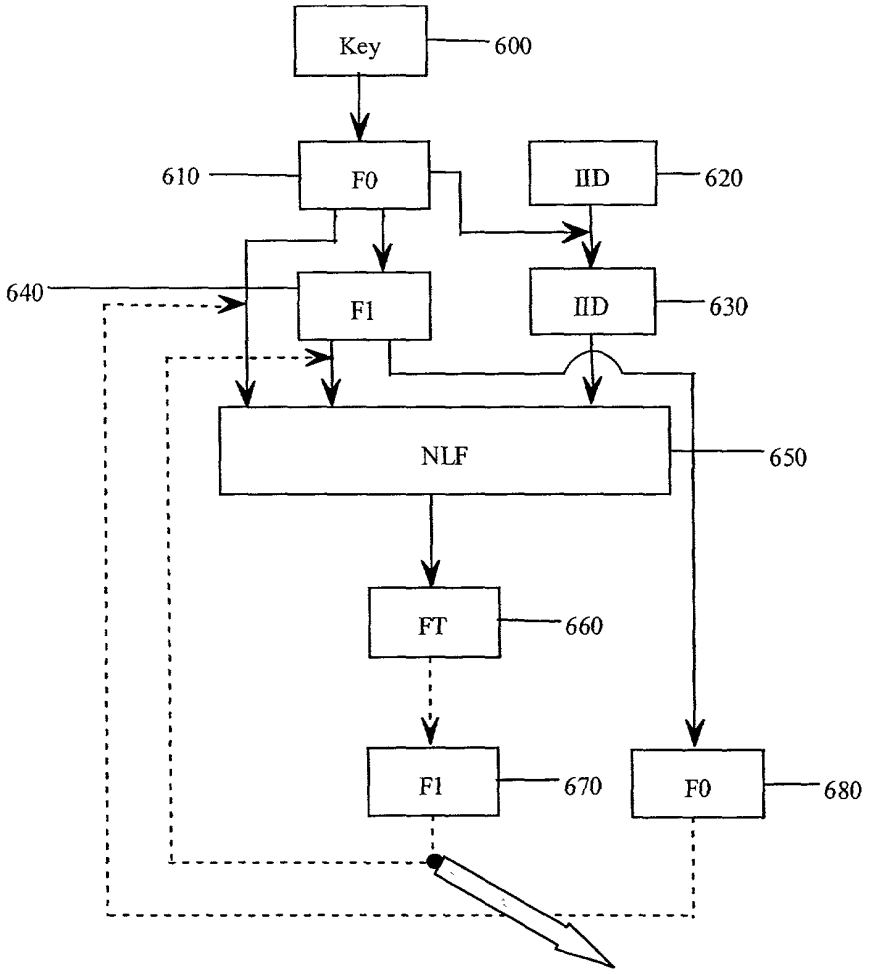


Figure 11A

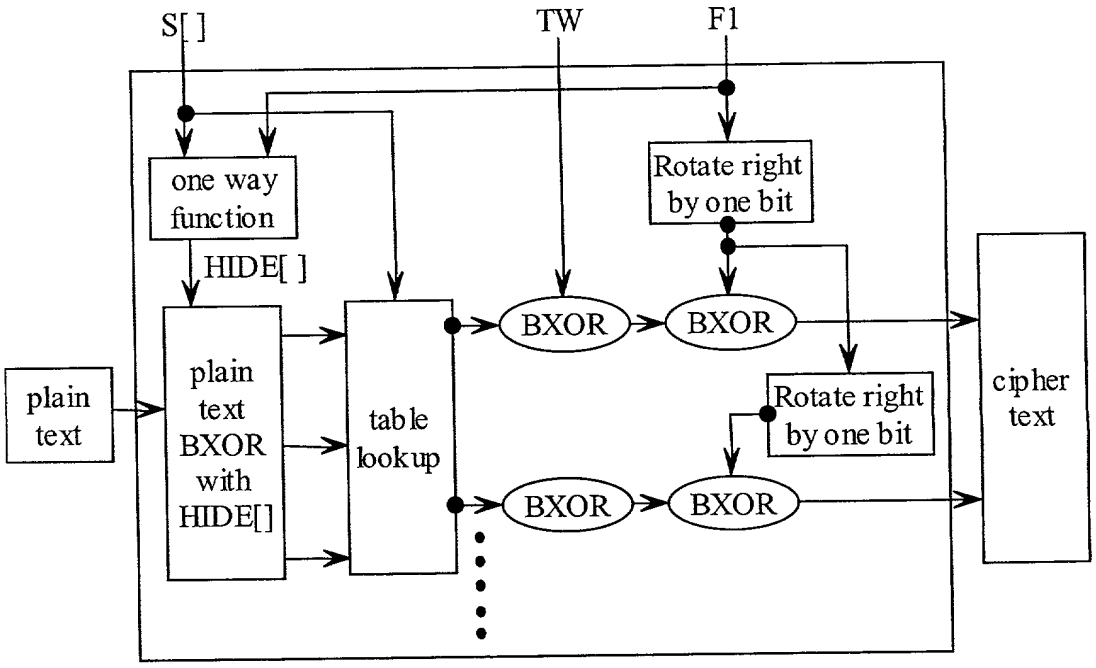

S[ ]
TW
F1
one way function
HIDE[ ]
Rotate right by one bit
plain text
plain text BXOR with HIDE[]
table lookup
BXOR
BXOR
Rotate right by one bit
BXOR
BXOR
cipher text

Figure 11B

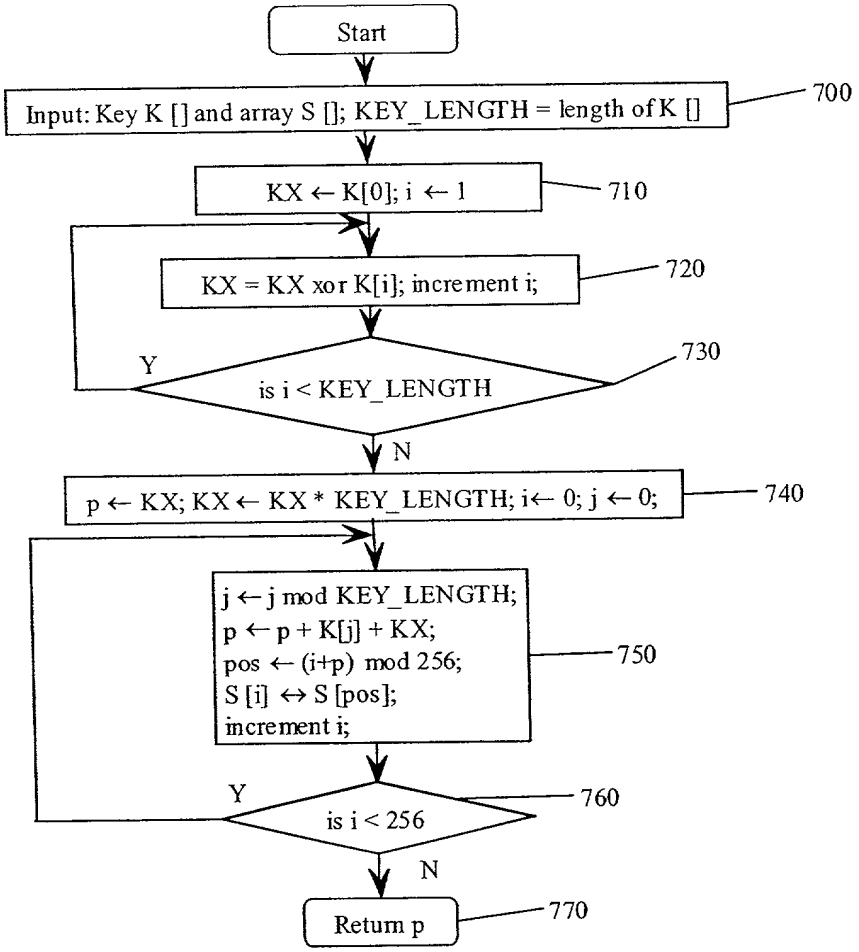

Start
Input: Key K [] and array S []; KEY_LENGTH = length of K []
700
KX ← K[0]; i ← 1
710
KX = KX xor K[i]; increment i;
720
Y
is i < KEY_LENGTH
730
N
p ← KX; KX ← KX * KEY_LENGTH; i← 0; j ← 0;
740
j ← j mod KEY_LENGTH;
p ← p + K[j] + KX;
pos ← (i+p) mod 256;
S [i] ↔ S [pos];
increment i;
750
Y
is i < 256
760
N
Return p
770

Figure 12

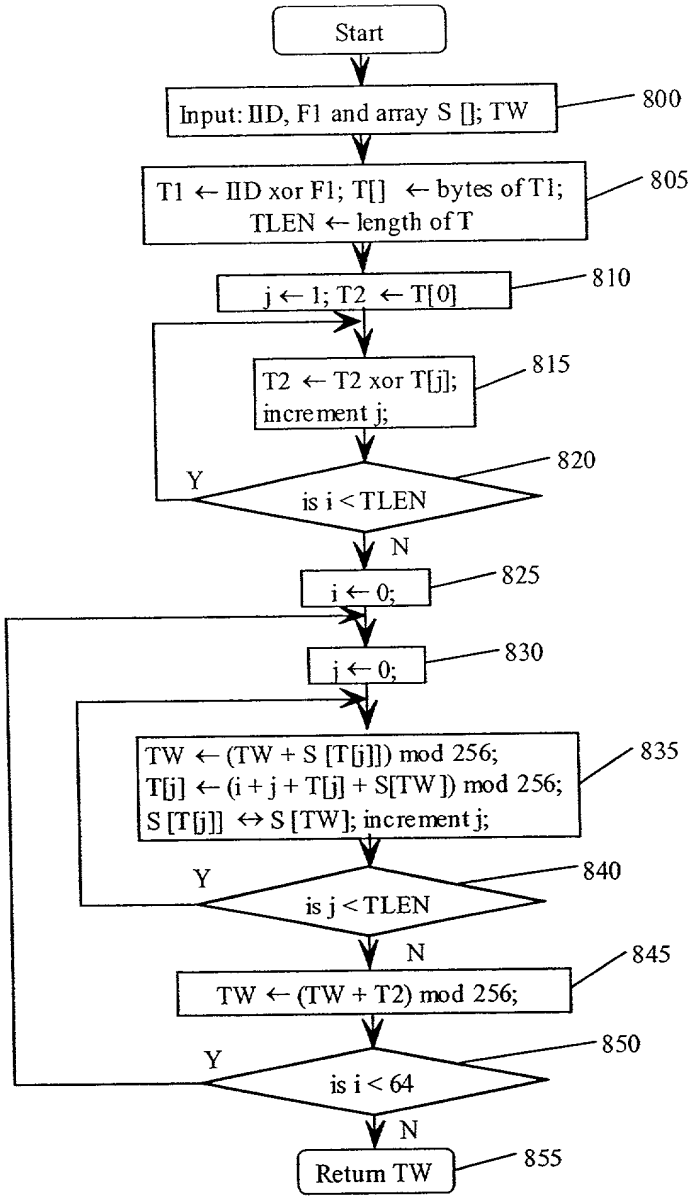

Start
Input: IID, F1 and array S []; TW
800
T1 ← IID xor F1; T[] ← bytes of T1;
TLEN ← length of T
805
j ← 1; T2 ← T[0]
810
T2 ← T2 xor T[j];
increment j;
815
Y
is i < TLEN
820
N
i ← 0;
825
j ← 0;
830
TW ← (TW + S [T[j]]) mod 256;
T[j] ← (i + j + T[j] + S[TW]) mod 256;
S [T[j]] ↔ S [TW]; increment j;
835
Y
is j < TLEN
840
N
TW ← (TW + T2) mod 256;
845
Y
is i < 64
850
N
Return TW
855

Figure 13

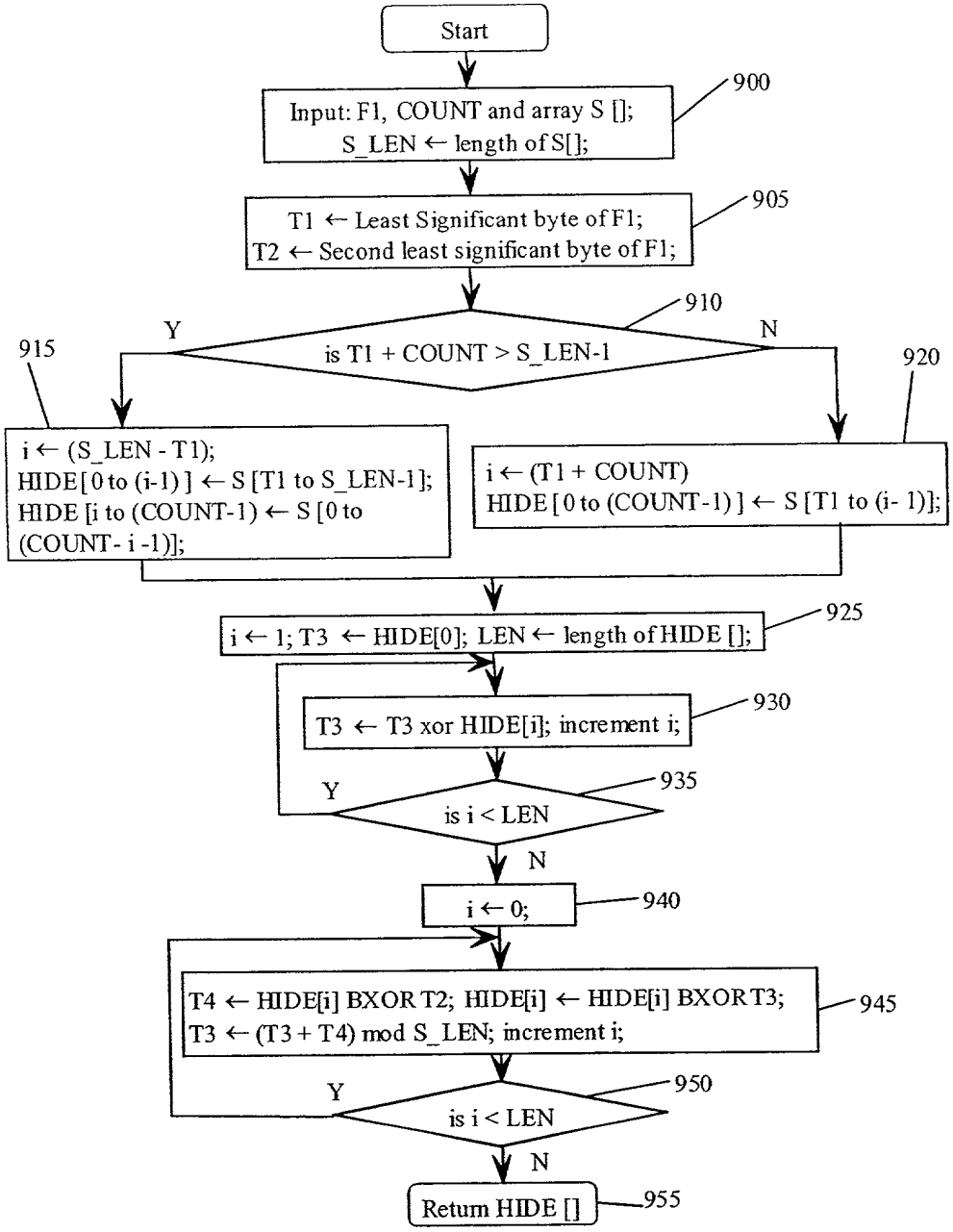

Start
Input: F1, COUNT and array S [];
S_LEN ← length of S[];
900
T1 ← Least Significant byte of F1;
T2 ← Second least significant byte of F1;
905
910
is T1 + COUNT > S_LEN-1
Y
N
915
920
i ← (S_LEN - T1);
HIDE [0 to (i-1)] ← S [T1 to S_LEN-1];
HIDE [i to (COUNT-1) ← S [0 to
(COUNT- i -1)];
i ← (T1 + COUNT)
HIDE [0 to (COUNT-1)] ← S [T1 to (i- 1)];
i ← 1; T3 ← HIDE[0]; LEN ← length of HIDE [];
925
T3 ← T3 xor HIDE[i]; increment i;
930
Y
is i < LEN
935
N
i ← 0;
940
T4 ← HIDE[i] BXOR T2; HIDE[i] ← HIDE[i] BXOR T3;
T3 ← (T3 + T4) mod S_LEN; increment i;
945
Y
is i < LEN
950
N
Retum HIDE []
955

Figure 14

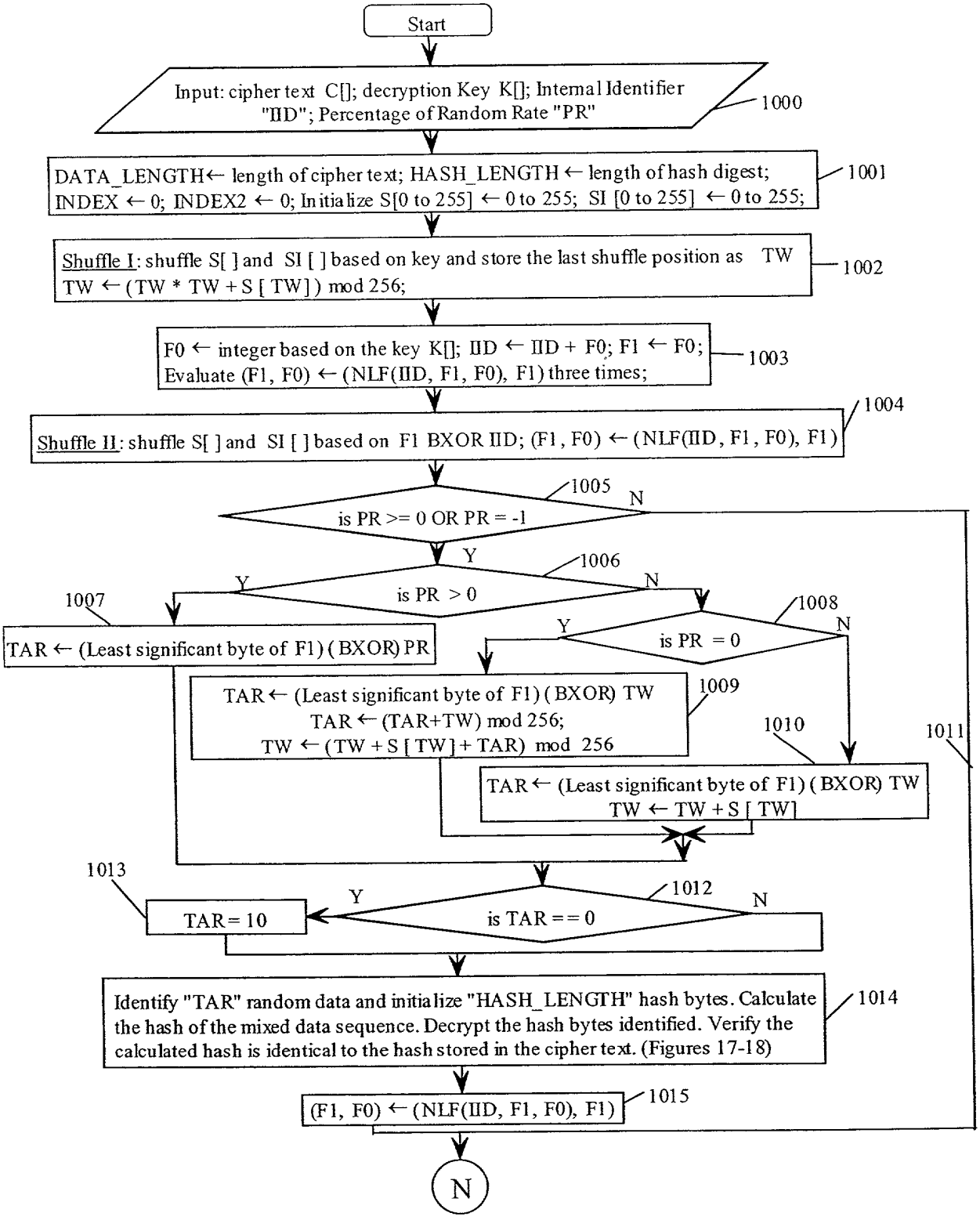

Start
Input: cipher text C[]; decryption Key K[]; Internal Identifier "IID"; Percentage of Random Rate "PR"
1000
DATA_LENGTH← length of cipher text; HASH_LENGTH ← length of hash digest; INDEX ← 0; INDEX2 ← 0; Initialize S[0 to 255] ← 0 to 255; SI [0 to 255] ← 0 to 255;
1001
Shuffle I: shuffle S[ ] and SI [ ] based on key and store the last shuffle position as TW
TW ← (TW * TW + S [TW]) mod 256;
1002
F0 ← integer based on the key K[]; IID ← IID + F0; F1 ← F0;
Evaluate (F1, F0) ← (NLF(IID, F1, F0), F1) three times;
1003
Shuffle II: shuffle S[ ] and SI [ ] based on F1 BXOR IID; (F1, F0) ← (NLF(IID, F1, F0), F1)
1004
is PR >= 0 OR PR = -1
1005
N
Y
is PR > 0
1006
Y
N
1007
TAR ← (Least significant byte of F1) (BXOR) PR
is PR = 0
1008
Y
N
TAR ← (Least significant byte of F1) (BXOR) TW
TAR ← (TAR+TW) mod 256;
TW ← (TW + S [TW] + TAR) mod 256
1009
1010
1011
TAR ← (Least significant byte of F1) (BXOR) TW
TW ← TW + S [TW]
1013
TAR = 10
Y
is TAR = = 0
1012
N
Identify "TAR" random data and initialize "HASH_LENGTH" hash bytes. Calculate the hash of the mixed data sequence. Decrypt the hash bytes identified. Verify the calculated hash is identical to the hash stored in the cipher text. (Figures 17-18)
1014
(F1, F0) ← (NLF(IID, F1, F0), F1)
1015
N

Figure 15

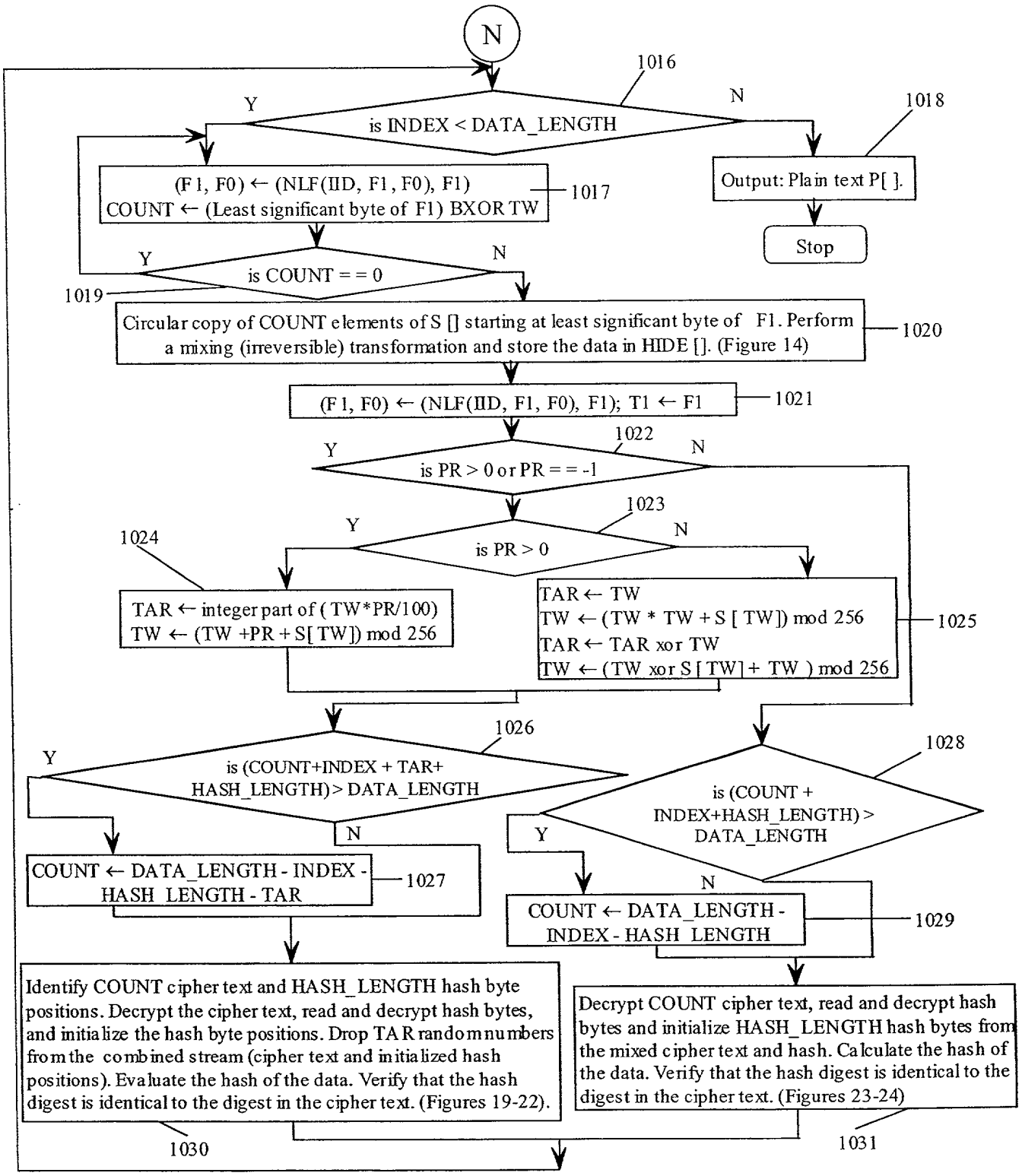

N
1016
is INDEX < DATA_LENGTH
Y
N
1018
Output: Plain text P[ ].
Stop
(F1, F0) ← (NLF(IID, F1, F0), F1)
COUNT ← (Least significant byte of F1) BXOR TW
1017
is COUNT = = 0
1019
Circular copy of COUNT elements of S [] starting at least significant byte of F1. Perform a mixing (irreversible) transformation and store the data in HIDE []. (Figure 14)
1020
(F1, F0) ← (NLF(IID, F1, F0), F1); T1 ← F1
1021
1022
is PR > 0 or PR = = -1
1023
is PR > 0
1024
TAR ← integer part of (TW*PR/100)
TW ← (TW +PR + S[TW]) mod 256
TAR ← TW
TW ← (TW * TW + S [TW]) mod 256
TAR ← TAR xor TW
TW ← (TW xor S [TW] + TW) mod 256
1025
1026
is (COUNT+INDEX + TAR+ HASH_LENGTH)> DATA_LENGTH
1028
is (COUNT + INDEX+HASH_LENGTH) > DATA_LENGTH
COUNT ← DATA_LENGTH - INDEX - HASH_LENGTH - TAR
1027
COUNT ← DATA_LENGTH - INDEX - HASH_LENGTH
1029
Identify COUNT cipher text and HASH_LENGTH hash byte positions. Decrypt the cipher text, read and decrypt hash bytes, and initialize the hash byte positions. Drop TAR random numbers from the combined stream (cipher text and initialized hash positions). Evaluate the hash of the data. Verify that the hash digest is identical to the digest in the cipher text. (Figures 19-22).
1030
Decrypt COUNT cipher text, read and decrypt hash bytes and initialize HASH_LENGTH hash bytes from the mixed cipher text and hash. Calculate the hash of the data. Verify that the hash digest is identical to the digest in the cipher text. (Figures 23-24)
1031

Figure 16

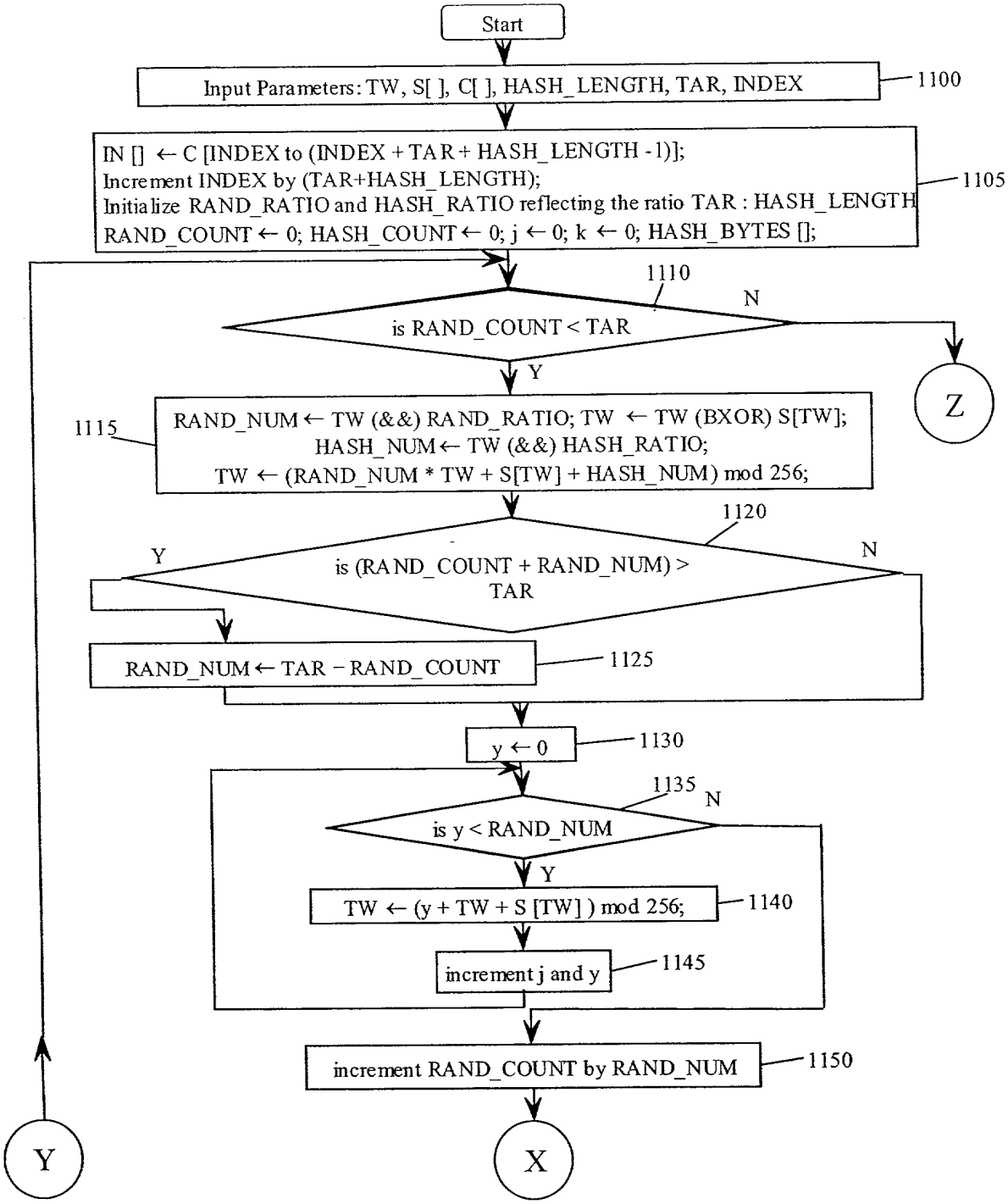

Start
Input Parameters: TW, S[ ], C[ ], HASH_LENGTH, TAR, INDEX
1100
IN [] ← C [INDEX to (INDEX + TAR + HASH_LENGTH -1)];
Increment INDEX by (TAR+HASH_LENGTH);
Initialize RAND_RATIO and HASH_RATIO reflecting the ratio TAR : HASH_LENGTH
RAND_COUNT ← 0; HASH_COUNT ← 0; j ← 0; k ← 0; HASH_BYTES [];
1105
1110
is RAND_COUNT < TAR
N
Y
Z
1115
RAND_NUM ← TW (&&) RAND_RATIO; TW ← TW (BXOR) S[TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (RAND_NUM * TW + S[TW] + HASH_NUM) mod 256;
1120
Y
is (RAND_COUNT + RAND_NUM) > TAR
N
RAND_NUM ← TAR − RAND_COUNT
1125
y ← 0
1130
1135
is y < RAND_NUM
N
Y
TW ← (y + TW + S [TW]) mod 256;
1140
increment j and y
1145
increment RAND_COUNT by RAND_NUM
1150
Y
X

Figure 17

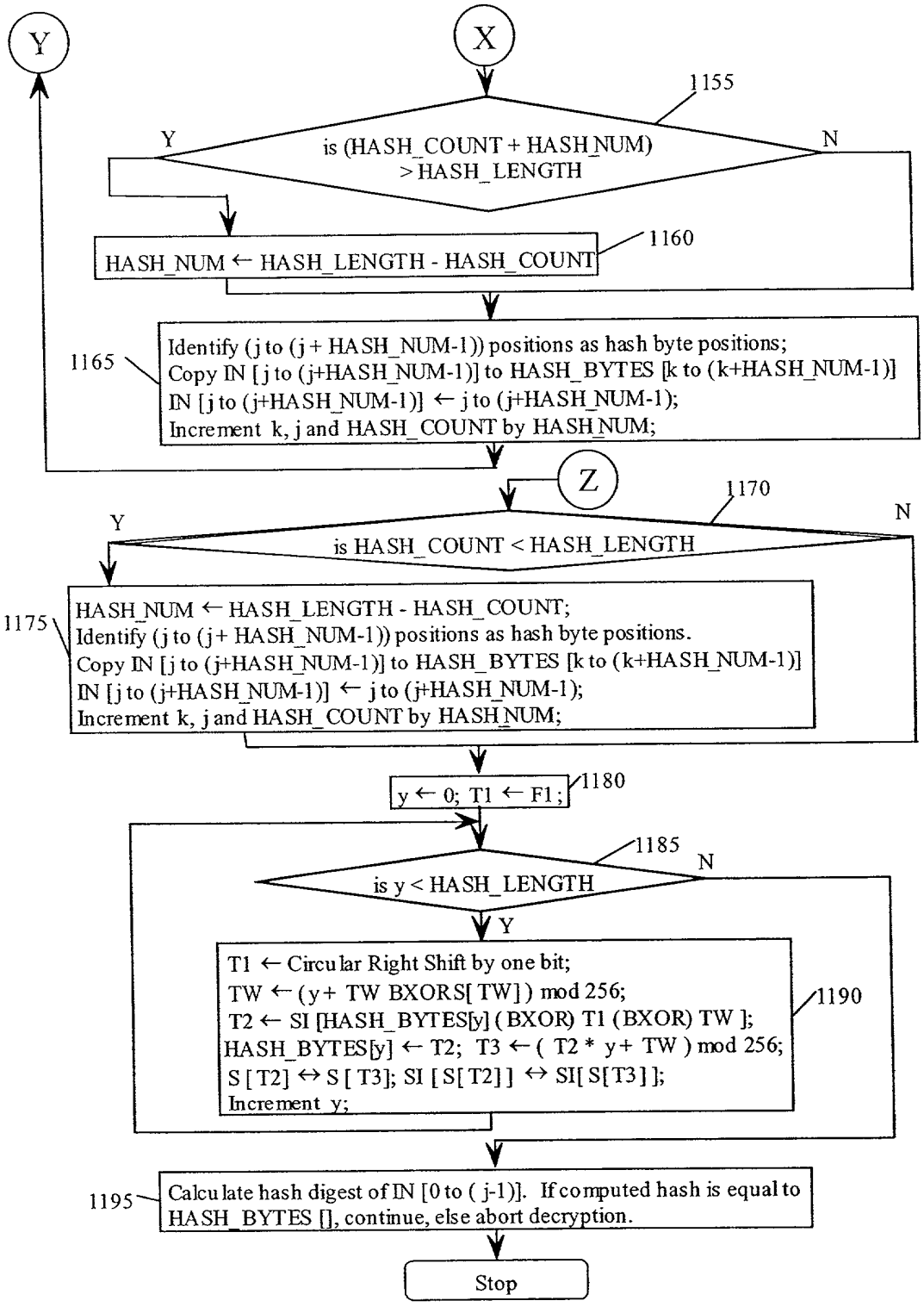

Y
X
1155
Y
is (HASH_COUNT + HASH_NUM)
> HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
1160
1165
Identify (j to (j + HASH_NUM-1)) positions as hash byte positions;
Copy IN [j to (j+HASH_NUM-1)] to HASH_BYTES [k to (k+HASH_NUM-1)]
IN [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1);
Increment k, j and HASH_COUNT by HASH_NUM;
Z
1170
Y
N
is HASH_COUNT < HASH_LENGTH
1175
HASH_NUM ← HASH_LENGTH - HASH_COUNT;
Identify (j to (j+ HASH_NUM-1)) positions as hash byte positions.
Copy IN [j to (j+HASH_NUM-1)] to HASH_BYTES [k to (k+HASH_NUM-1)]
IN [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1);
Increment k, j and HASH_COUNT by HASH_NUM;
y ← 0; T1 ← F1;
1180
1185
N
is y < HASH_LENGTH
Y
T1 ← Circular Right Shift by one bit;
TW ← (y+ TW BXORS[TW]) mod 256;
T2 ← SI [HASH_BYTES[y] (BXOR) T1 (BXOR) TW];
HASH_BYTES[y] ← T2; T3 ← (T2 * y+ TW) mod 256;
S [T2] ↔ S [T3]; SI [S[T2]] ↔ SI[S[T3]];
Increment y;
1190
1195
Calculate hash digest of IN [0 to (j-1)]. If computed hash is equal to
HASH_BYTES [], continue, else abort decryption.
Stop

Figure 18

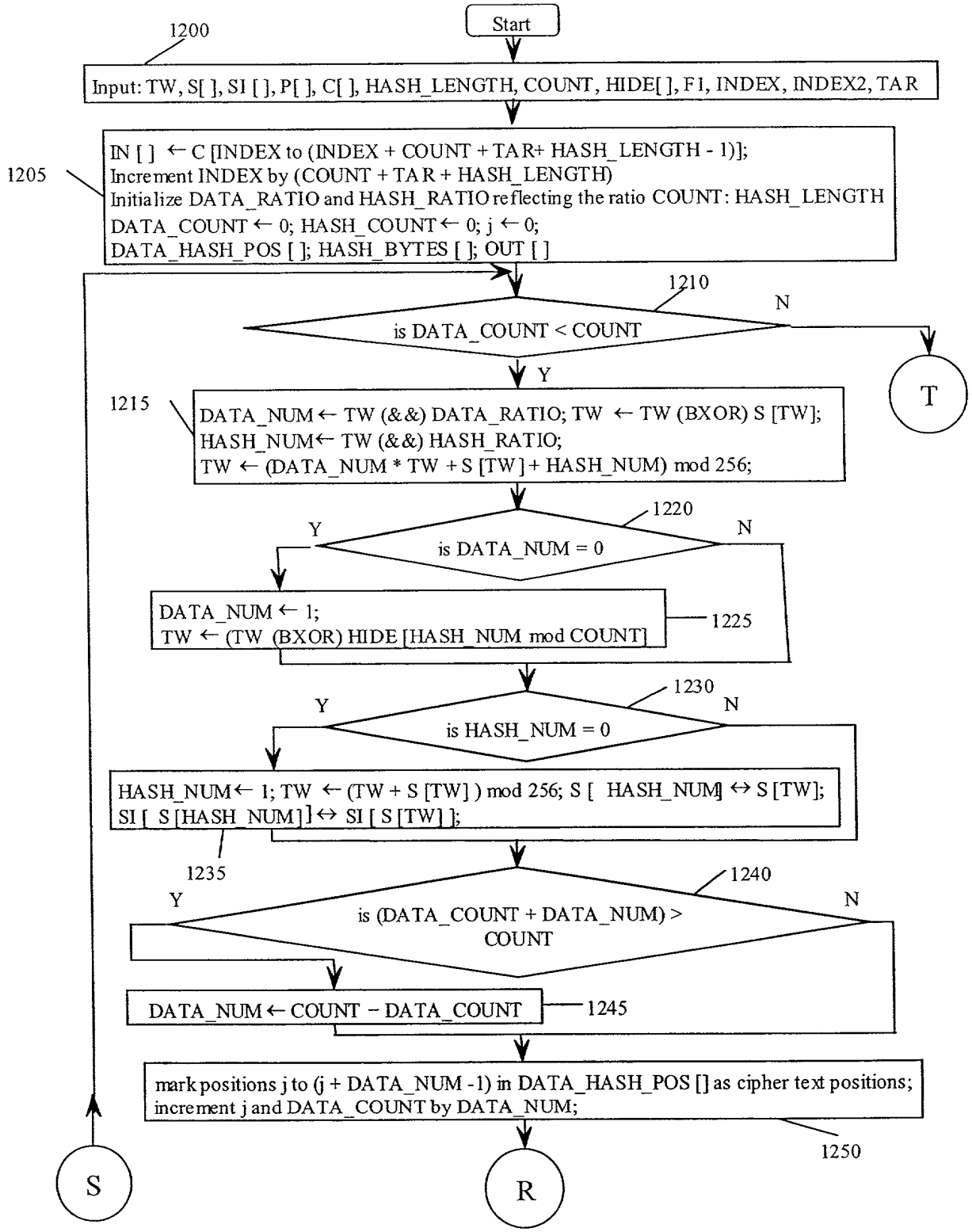

Start
1200
Input: TW, S[ ], SI [ ], P[ ], C[ ], HASH_LENGTH, COUNT, HIDE[ ], F1, INDEX, INDEX2, TAR
1205
IN [ ] ← C [INDEX to (INDEX + COUNT + TAR+ HASH_LENGTH - 1)];
Increment INDEX by (COUNT + TAR + HASH_LENGTH)
Initialize DATA_RATIO and HASH_RATIO reflecting the ratio COUNT: HASH_LENGTH
DATA_COUNT ← 0; HASH_COUNT ← 0; j ← 0;
DATA_HASH_POS [ ]; HASH_BYTES [ ]; OUT [ ]
1210
is DATA_COUNT < COUNT
N
Y
T
1215
DATA_NUM ← TW (&&) DATA_RATIO; TW ← TW (BXOR) S [TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (DATA_NUM * TW + S [TW] + HASH_NUM) mod 256;
1220
Y
is DATA_NUM = 0
N
DATA_NUM ← 1;
TW ← (TW (BXOR) HIDE [HASH_NUM mod COUNT]
1225
1230
Y
is HASH_NUM = 0
N
HASH_NUM ← 1; TW ← (TW + S [TW]) mod 256; S [HASH_NUM] ↔ S [TW];
SI [S [HASH_NUM]] ↔ SI [S [TW]];
1235
1240
Y
is (DATA_COUNT + DATA_NUM) > COUNT
N
DATA_NUM ← COUNT − DATA_COUNT
1245
mark positions j to (j + DATA_NUM -1) in DATA_HASH_POS [] as cipher text positions;
increment j and DATA_COUNT by DATA_NUM;
1250
S
R

Figure 19

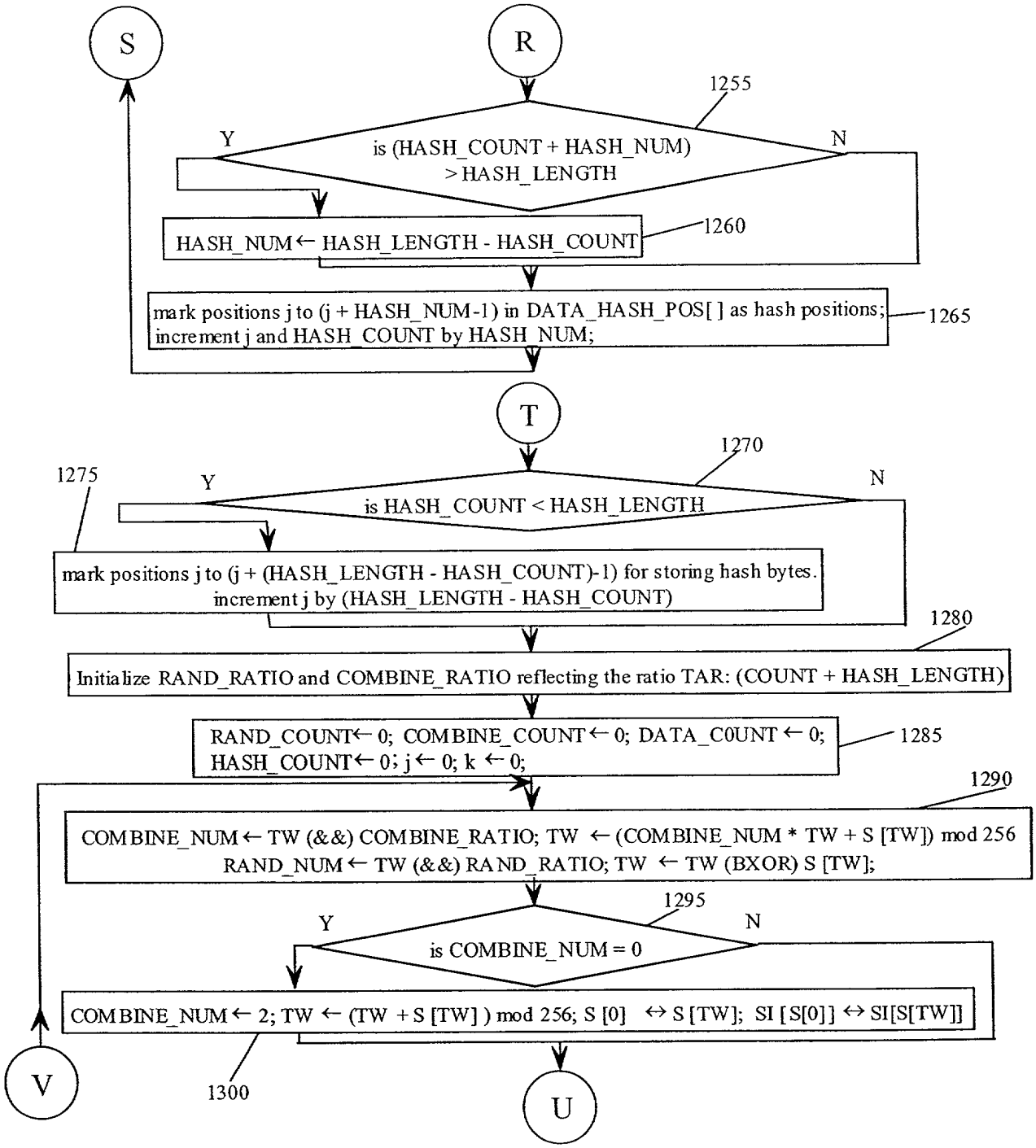

S
R
1255
Y
is (HASH_COUNT + HASH_NUM) > HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
1260
mark positions j to (j + HASH_NUM-1) in DATA_HASH_POS[ ] as hash positions;
increment j and HASH_COUNT by HASH_NUM;
1265
T
1270
1275
Y
is HASH_COUNT < HASH_LENGTH
N
mark positions j to (j + (HASH_LENGTH - HASH_COUNT)-1) for storing hash bytes.
increment j by (HASH_LENGTH - HASH_COUNT)
1280
Initialize RAND_RATIO and COMBINE_RATIO reflecting the ratio TAR: (COUNT + HASH_LENGTH)
RAND_COUNT ← 0; COMBINE_COUNT ← 0; DATA_C0UNT ← 0;
HASH_COUNT ← 0; j ← 0; k ← 0;
1285
1290
COMBINE_NUM ← TW (&&) COMBINE_RATIO; TW ← (COMBINE_NUM * TW + S [TW]) mod 256
RAND_NUM ← TW (&&) RAND_RATIO; TW ← TW (BXOR) S [TW];
1295
Y
is COMBINE_NUM = 0
N
COMBINE_NUM ← 2; TW ← (TW + S [TW]) mod 256; S [0] ↔ S [TW]; SI [S[0]] ↔ SI[S[TW]]
V
1300
U

Figure 20

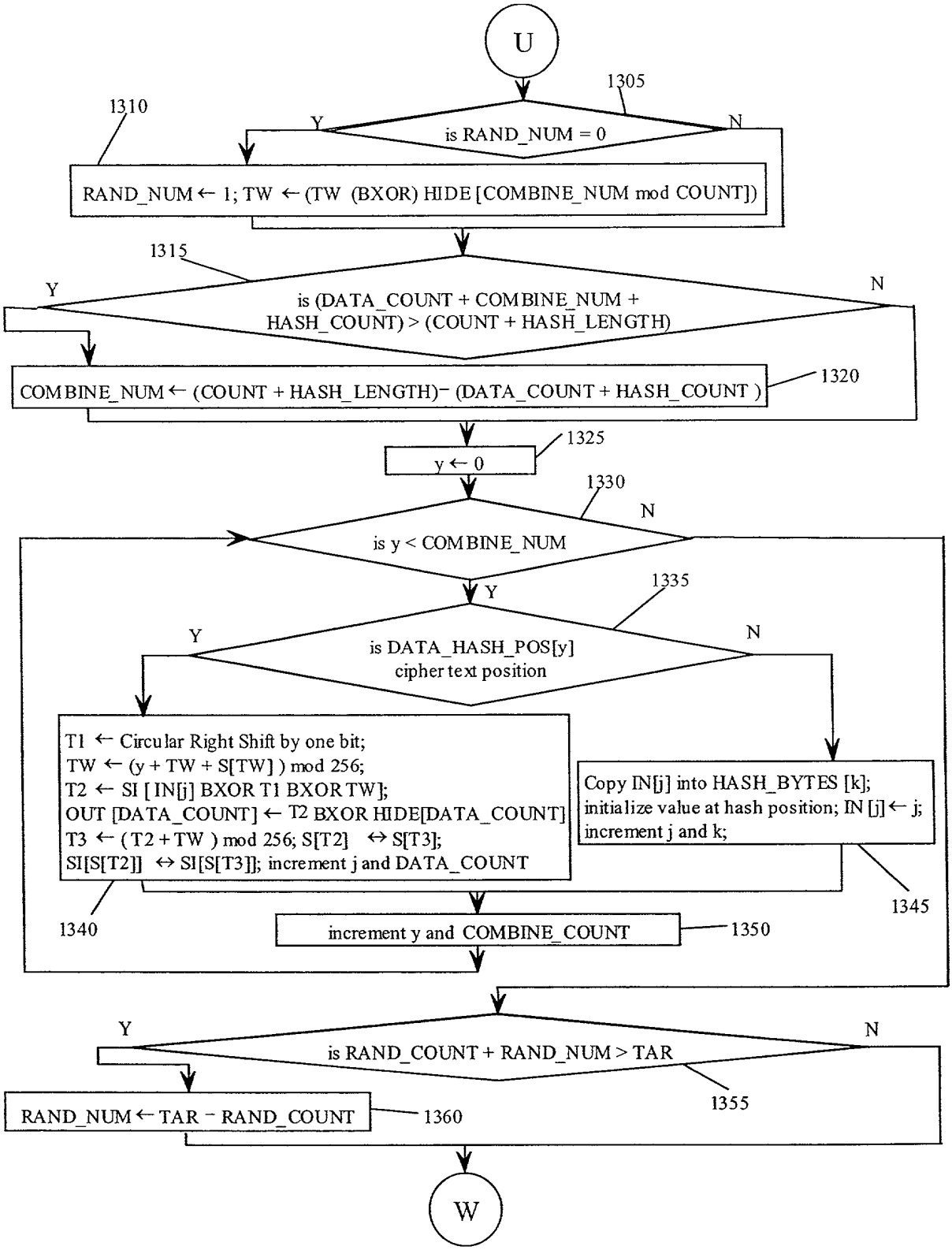

U
1305
is RAND_NUM = 0
Y
N
1310
RAND_NUM ← 1; TW ← (TW (BXOR) HIDE [COMBINE_NUM mod COUNT])
1315
is (DATA_COUNT + COMBINE_NUM + HASH_COUNT) > (COUNT + HASH_LENGTH)
COMBINE_NUM ← (COUNT + HASH_LENGTH)− (DATA_COUNT + HASH_COUNT)
1320
y ← 0
1325
1330
is y < COMBINE_NUM
1335
is DATA_HASH_POS[y] cipher text position
T1 ← Circular Right Shift by one bit;
TW ← (y + TW + S[TW]) mod 256;
T2 ← SI [IN[j] BXOR T1 BXOR TW];
OUT [DATA_COUNT] ← T2 BXOR HIDE[DATA_COUNT]
T3 ← (T2 + TW) mod 256; S[T2] ↔ S[T3];
SI[S[T2]] ↔ SI[S[T3]]; increment j and DATA_COUNT
1340
Copy IN[j] into HASH_BYTES [k];
initialize value at hash position; IN [j] ← j;
increment j and k;
1345
increment y and COMBINE_COUNT
1350
is RAND_COUNT + RAND_NUM > TAR
1355
RAND_NUM ← TAR − RAND_COUNT
1360
W

Figure 21

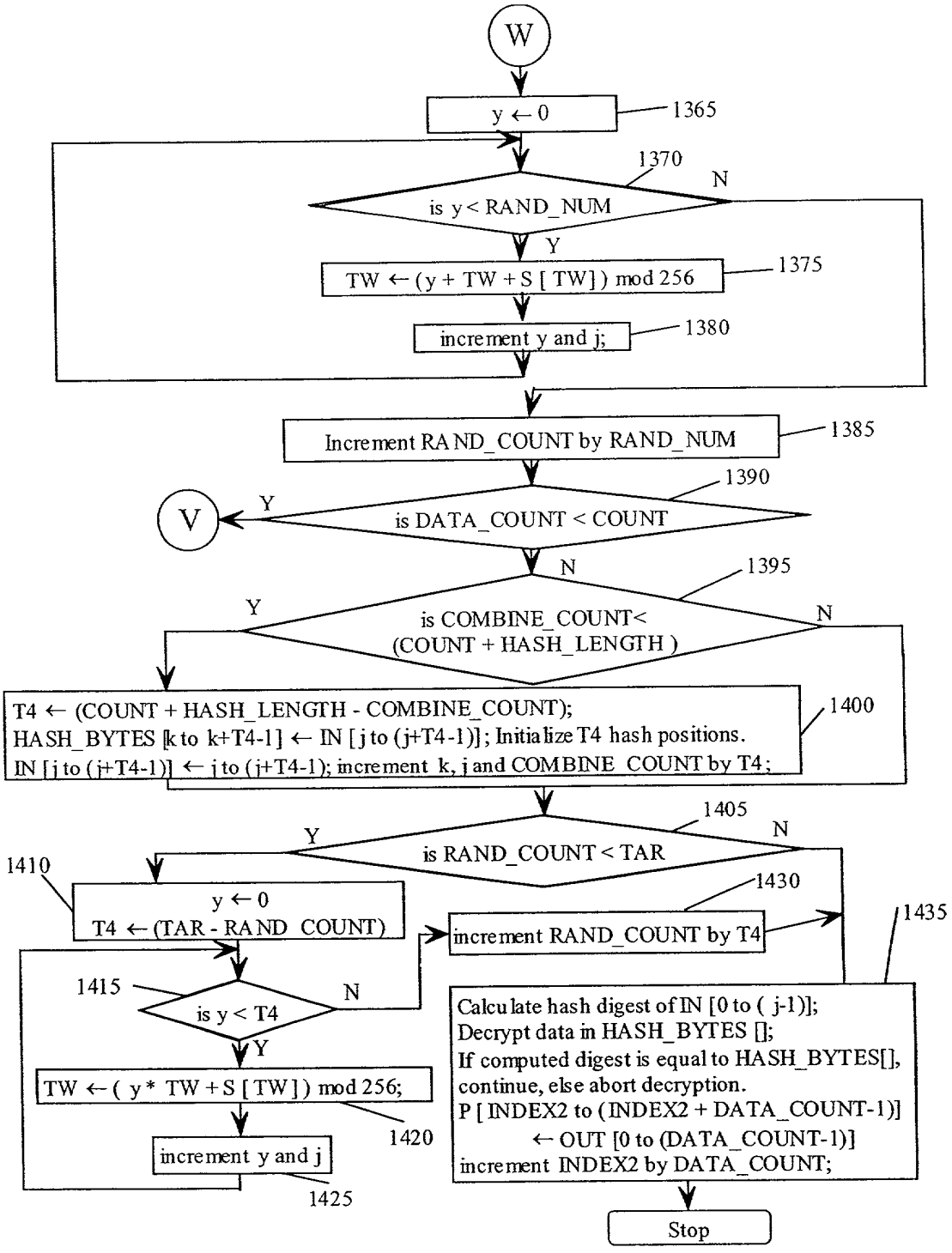

W
y ← 0
1365
1370
is y < RAND_NUM
N
Y
TW ← (y + TW + S [TW]) mod 256
1375
increment y and j;
1380
Increment RAND_COUNT by RAND_NUM
1385
1390
V
Y
is DATA_COUNT < COUNT
N
1395
Y
is COMBINE_COUNT<
(COUNT + HASH_LENGTH)
N
T4 ← (COUNT + HASH_LENGTH - COMBINE_COUNT);
HASH_BYTES [k to k+T4-1] ← IN [j to (j+T4-1)]; Initialize T4 hash positions.
IN [j to (j+T4-1)] ← j to (j+T4-1); increment k, j and COMBINE_COUNT by T4;
1400
1405
Y
is RAND_COUNT < TAR
N
1410
y ← 0
T4 ← (TAR - RAND_COUNT)
1430
increment RAND_COUNT by T4
1435
1415
is y < T4
N
Y
TW ← (y * TW + S [TW]) mod 256;
1420
increment y and j
1425
Calculate hash digest of IN [0 to (j-1)];
Decrypt data in HASH_BYTES [];
If computed digest is equal to HASH_BYTES[],
continue, else abort decryption.
P [INDEX2 to (INDEX2 + DATA_COUNT-1)]
← OUT [0 to (DATA_COUNT-1)]
increment INDEX2 by DATA_COUNT;
Stop

Figure 22

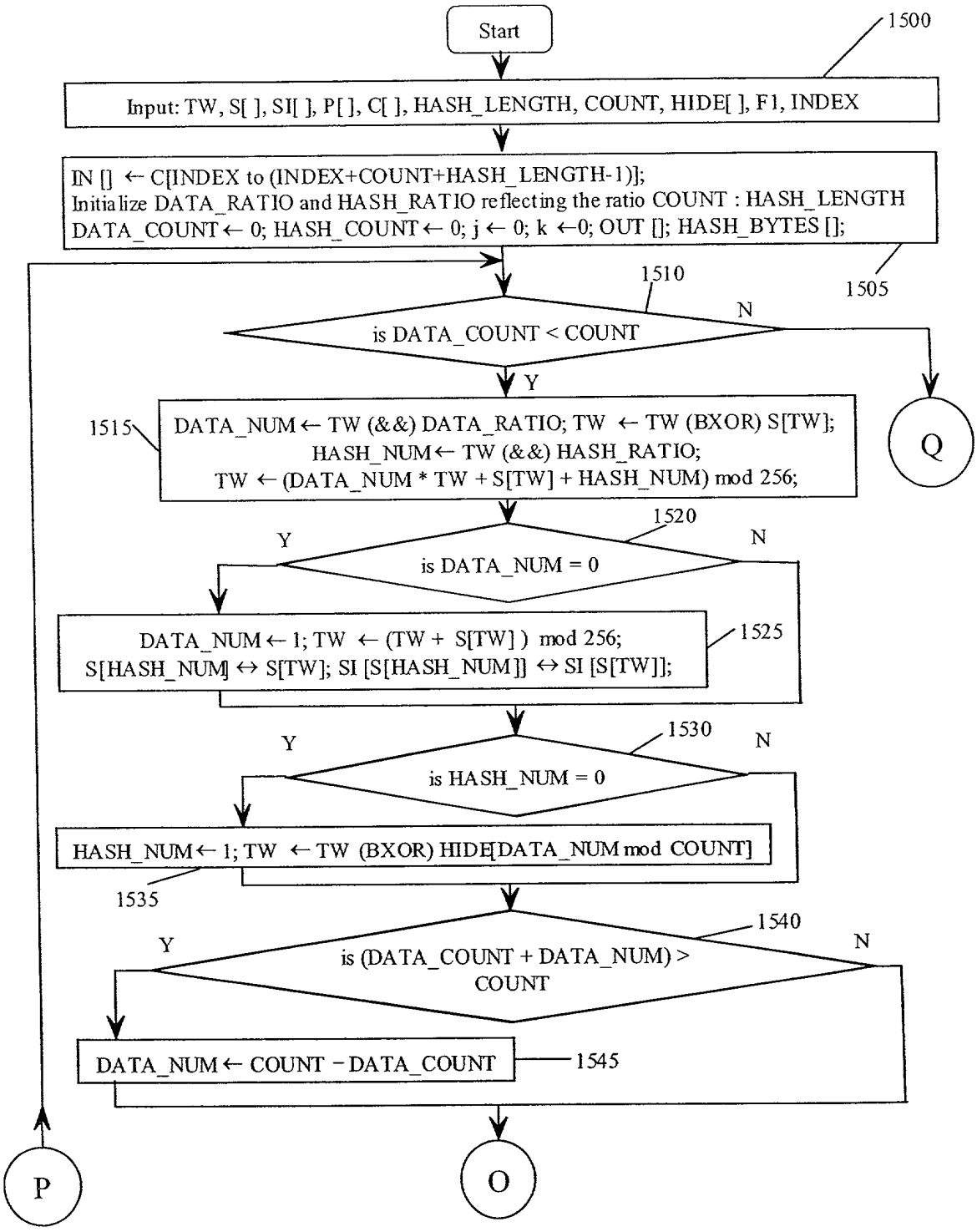

Start
1500
Input: TW, S[ ], SI[ ], P[ ], C[ ], HASH_LENGTH, COUNT, HIDE[ ], F1, INDEX
IN [] ← C[INDEX to (INDEX+COUNT+HASH_LENGTH-1)];
Initialize DATA_RATIO and HASH_RATIO reflecting the ratio COUNT : HASH_LENGTH
DATA_COUNT ← 0; HASH_COUNT ← 0; j ← 0; k ←0; OUT []; HASH_BYTES [];
1505
1510
is DATA_COUNT < COUNT
N
Y
Q
1515
DATA_NUM ← TW (&&) DATA_RATIO; TW ← TW (BXOR) S[TW];
HASH_NUM ← TW (&&) HASH_RATIO;
TW ← (DATA_NUM * TW + S[TW] + HASH_NUM) mod 256;
1520
Y
is DATA_NUM = 0
N
DATA_NUM ← 1; TW ← (TW + S[TW]) mod 256;
S[HASH_NUM] ↔ S[TW]; SI [S[HASH_NUM]] ↔ SI [S[TW]];
1525
1530
Y
is HASH_NUM = 0
N
HASH_NUM ← 1; TW ← TW (BXOR) HIDE[DATA_NUM mod COUNT]
1535
1540
Y
is (DATA_COUNT + DATA_NUM) >
COUNT
N
DATA_NUM ← COUNT − DATA_COUNT
1545
P
O

Figure 23

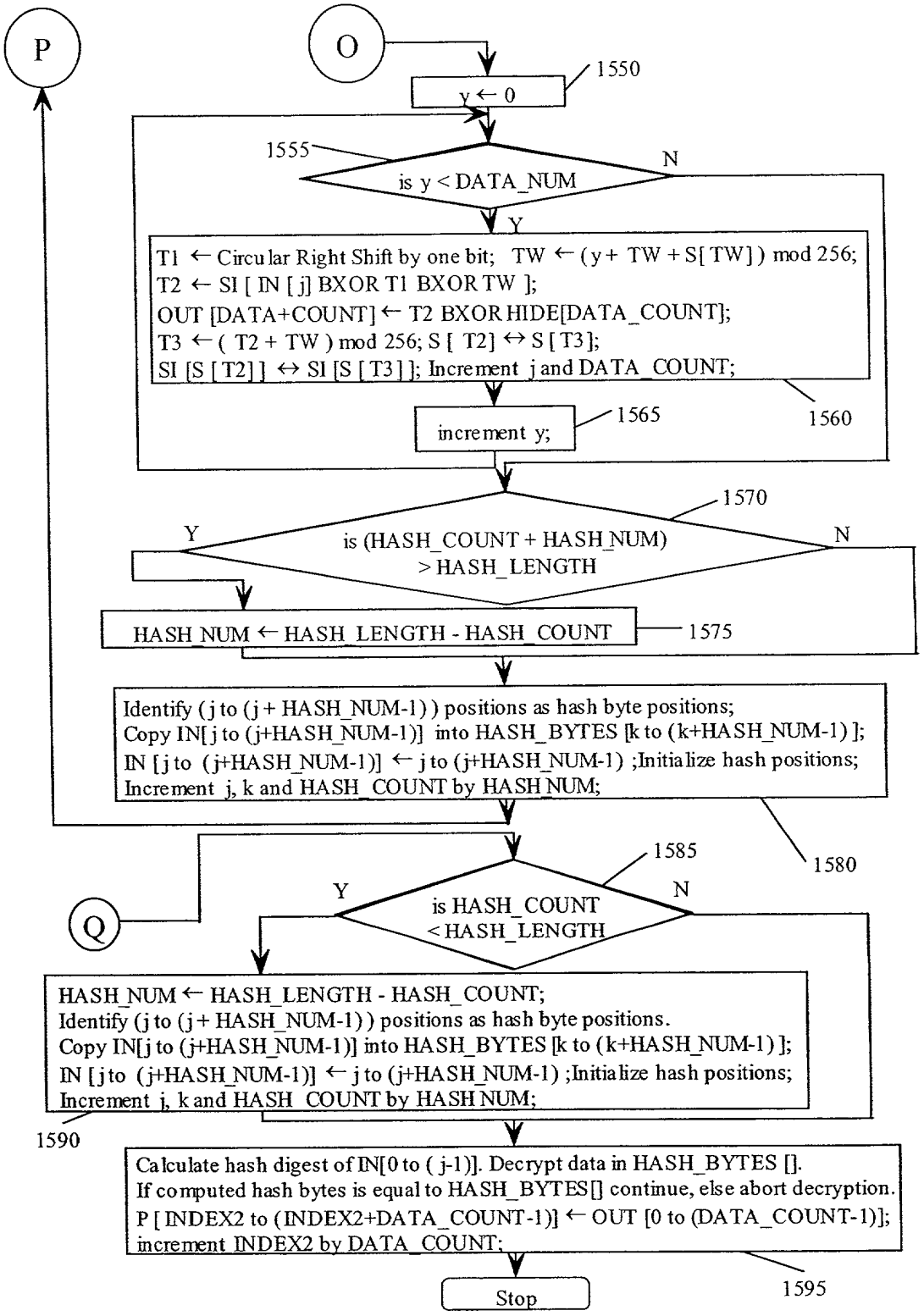

P
O
1550
y ← 0
1555
is y < DATA_NUM
N
Y
T1 ← Circular Right Shift by one bit; TW ← (y + TW + S[TW]) mod 256;
T2 ← SI [IN [j] BXOR T1 BXOR TW];
OUT [DATA+COUNT] ← T2 BXOR HIDE[DATA_COUNT];
T3 ← (T2 + TW) mod 256; S [T2] ↔ S [T3];
SI [S [T2]] ↔ SI [S [T3]]; Increment j and DATA_COUNT;
1560
increment y;
1565
1570
Y
is (HASH_COUNT + HASH_NUM)
> HASH_LENGTH
N
HASH_NUM ← HASH_LENGTH - HASH_COUNT
1575
Identify (j to (j + HASH_NUM-1)) positions as hash byte positions;
Copy IN[j to (j+HASH_NUM-1)] into HASH_BYTES [k to (k+HASH_NUM-1)];
IN [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1) ;Initialize hash positions;
Increment j, k and HASH_COUNT by HASH_NUM;
1580
1585
Y
is HASH_COUNT
< HASH_LENGTH
N
Q
HASH_NUM ← HASH_LENGTH - HASH_COUNT;
Identify (j to (j + HASH_NUM-1)) positions as hash byte positions.
Copy IN[j to (j+HASH_NUM-1)] into HASH_BYTES [k to (k+HASH_NUM-1)];
IN [j to (j+HASH_NUM-1)] ← j to (j+HASH_NUM-1) ;Initialize hash positions;
Increment j, k and HASH_COUNT by HASH_NUM;
1590
Calculate hash digest of IN[0 to (j-1)]. Decrypt data in HASH_BYTES [].
If computed hash bytes is equal to HASH_BYTES[] continue, else abort decryption.
P [INDEX2 to (INDEX2+DATA_COUNT-1)] ← OUT [0 to (DATA_COUNT-1)];
increment INDEX2 by DATA_COUNT;
1595
Stop

Figure 24

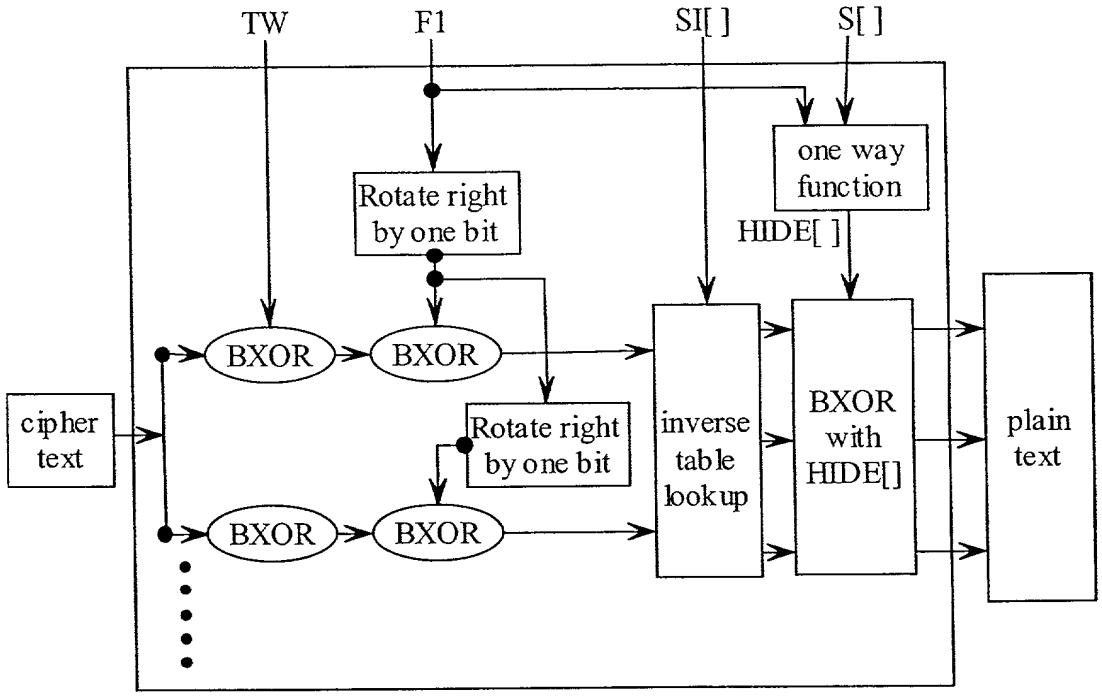

TW
F1
SI[ ]
S[ ]
one way function
Rotate right by one bit
HIDE[ ]
BXOR
BXOR
cipher text
Rotate right by one bit
inverse table lookup
BXOR with HIDE[]
plain text
BXOR
BXOR

Figure 25

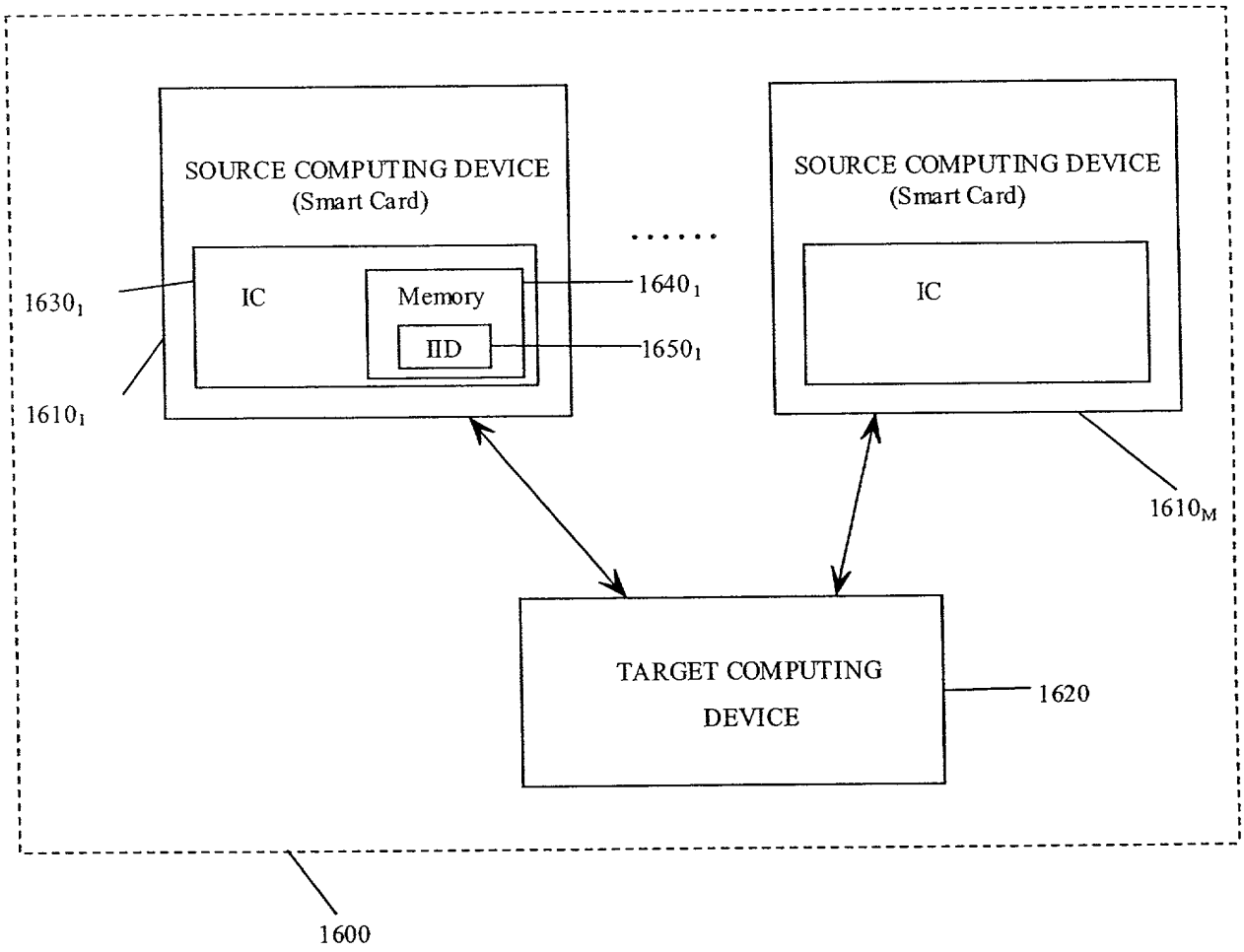

SOURCE COMPUTING DEVICE
(Smart Card)
IC
Memory
IID
1630₁
1610₁
1640₁
1650₁
SOURCE COMPUTING DEVICE
(Smart Card)
IC
1610M
TARGET COMPUTING
DEVICE
1620
1600

Figure 26

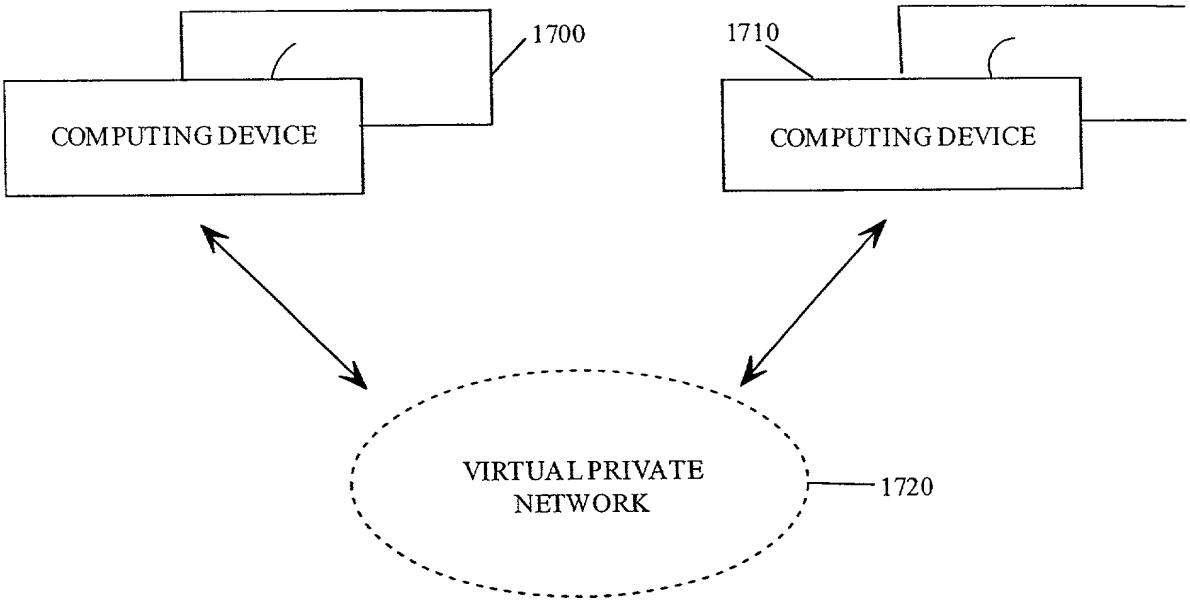

1700
COMPUTING DEVICE
1710
COMPUTING DEVICE
VIRTUAL PRIVATE
NETWORK
1720

Figure 27

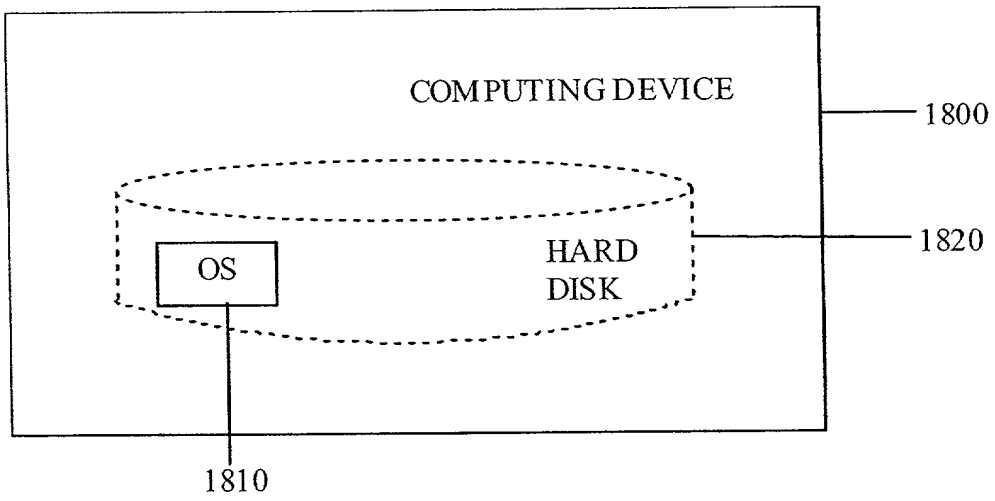


Figure 28

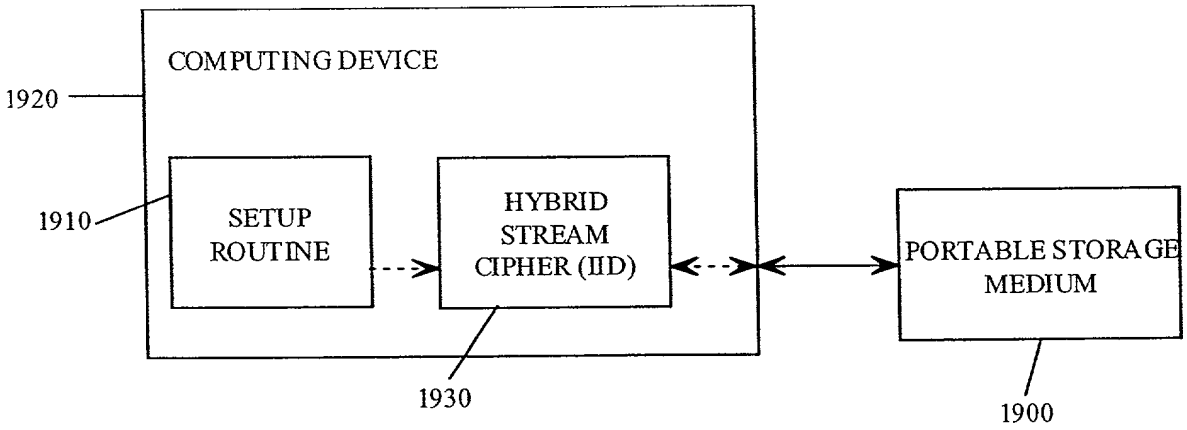


Figure 29

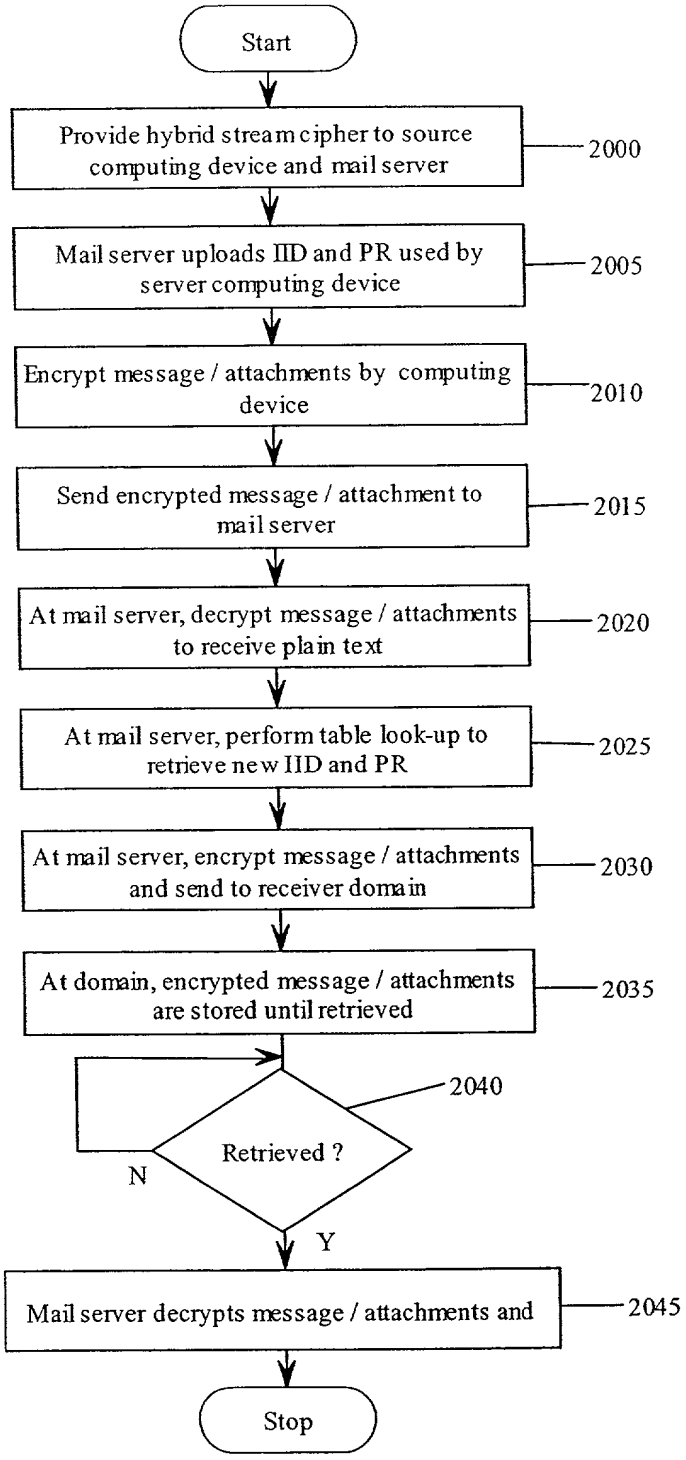

Start
Provide hybrid stream cipher to source computing device and mail server
2000
Mail server uploads IID and PR used by server computing device
2005
Encrypt message / attachments by computing device
2010
Send encrypted message / attachment to mail server
2015
At mail server, decrypt message / attachments to receive plain text
2020
At mail server, perform table look-up to retrieve new IID and PR
2025
At mail server, encrypt message / attachments and send to receiver domain
2030
At domain, encrypted message / attachments are stored until retrieved
2035
Retrieved ?
2040
N
Y
Mail server decrypts message / attachments and
2045
Stop

Figure 30

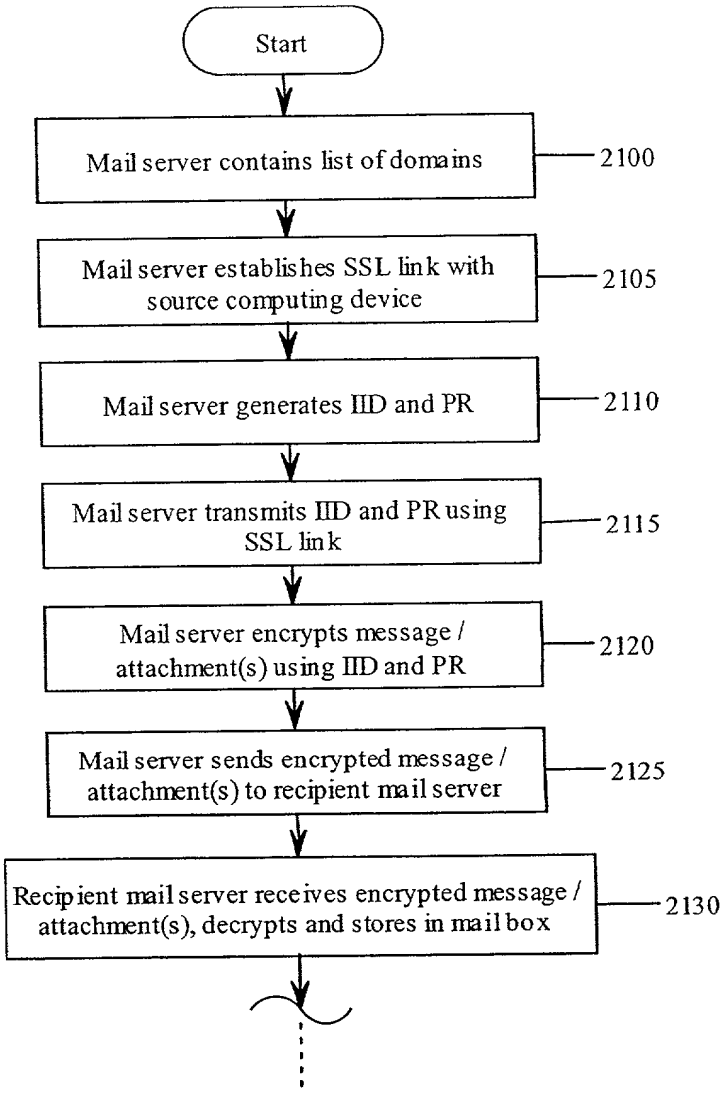

Start
Mail server contains list of domains
2100
Mail server establishes SSL link with source computing device
2105
Mail server generates IID and PR
2110
Mail server transmits IID and PR using SSL link
2115
Mail server encrypts message / attachment(s) using IID and PR
2120
Mail server sends encrypted message / attachment(s) to recipient mail server
2125
Recipient mail server receives encrypted message / attachment(s), decrypts and stores in mail box
2130

Figure 31

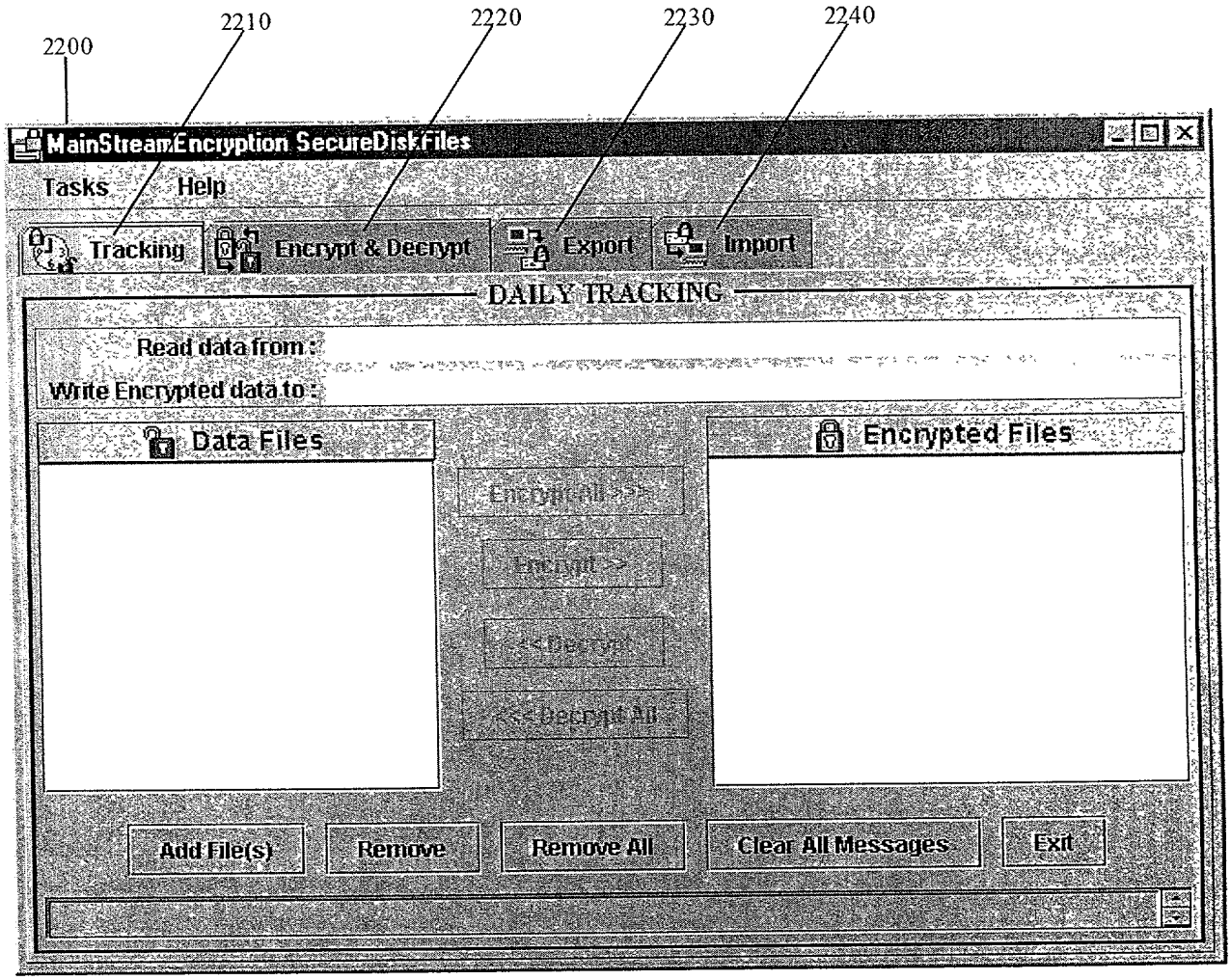

2200
2210
2220
2230
2240
MainStream Encryption SecureDisk Files
Tasks
Help
Tracking
Encrypt & Decrypt
Export
Import
DAILY TRACKING
Read data from :
Write Encrypted data to :
Data Files
Encrypted Files
Encrypt All >>
Encrypt >
< Decrypt
<< Decrypt All
Add File(s)
Remove
Remove All
Clear All Messages
Exit

Figure 32

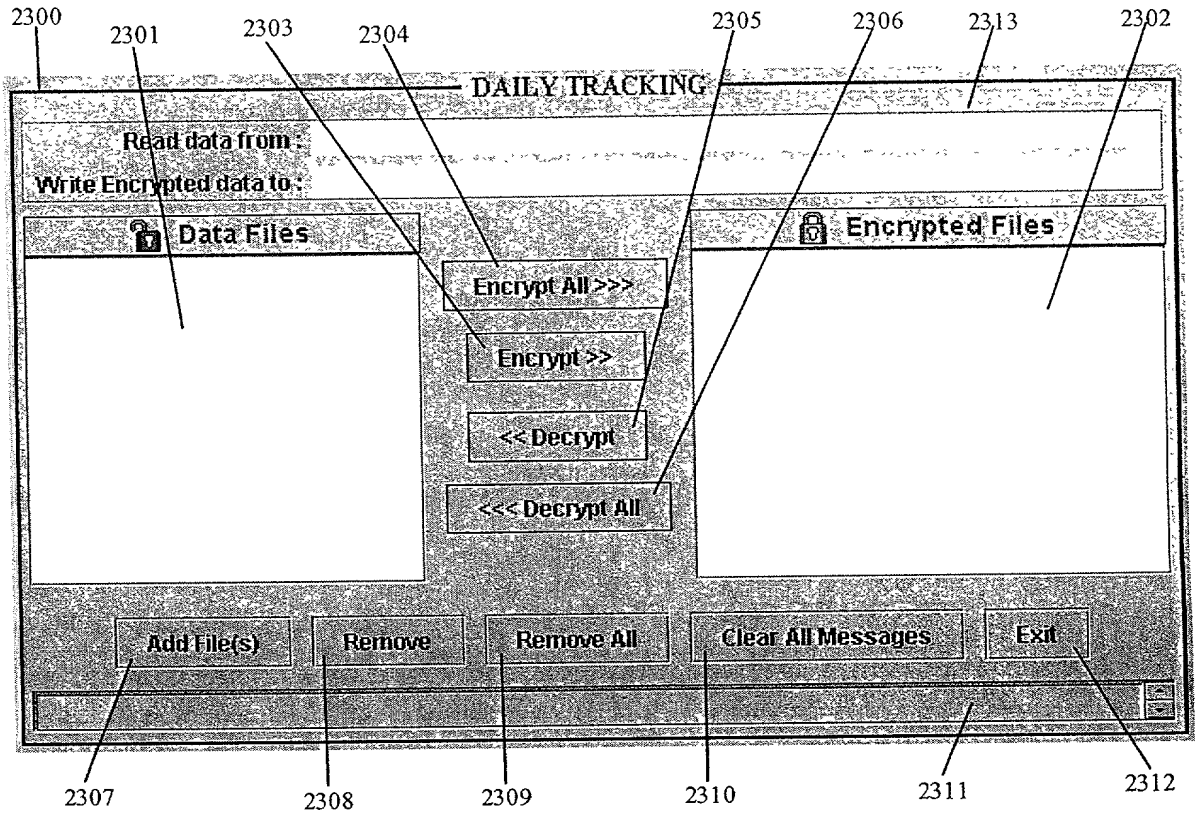

2300
2301
2303
2304
2305
2306
2313
2302
DAILY TRACKING
Read data from :
Write Encrypted data to :
Data Files
Encrypted Files
Encrypt All >>>
Encrypt >>
<< Decrypt
<<< Decrypt All
Add File(s)
Remove
Remove All
Clear All Messages
Exit
2307
2308
2309
2310
2311
2312

Figure 33

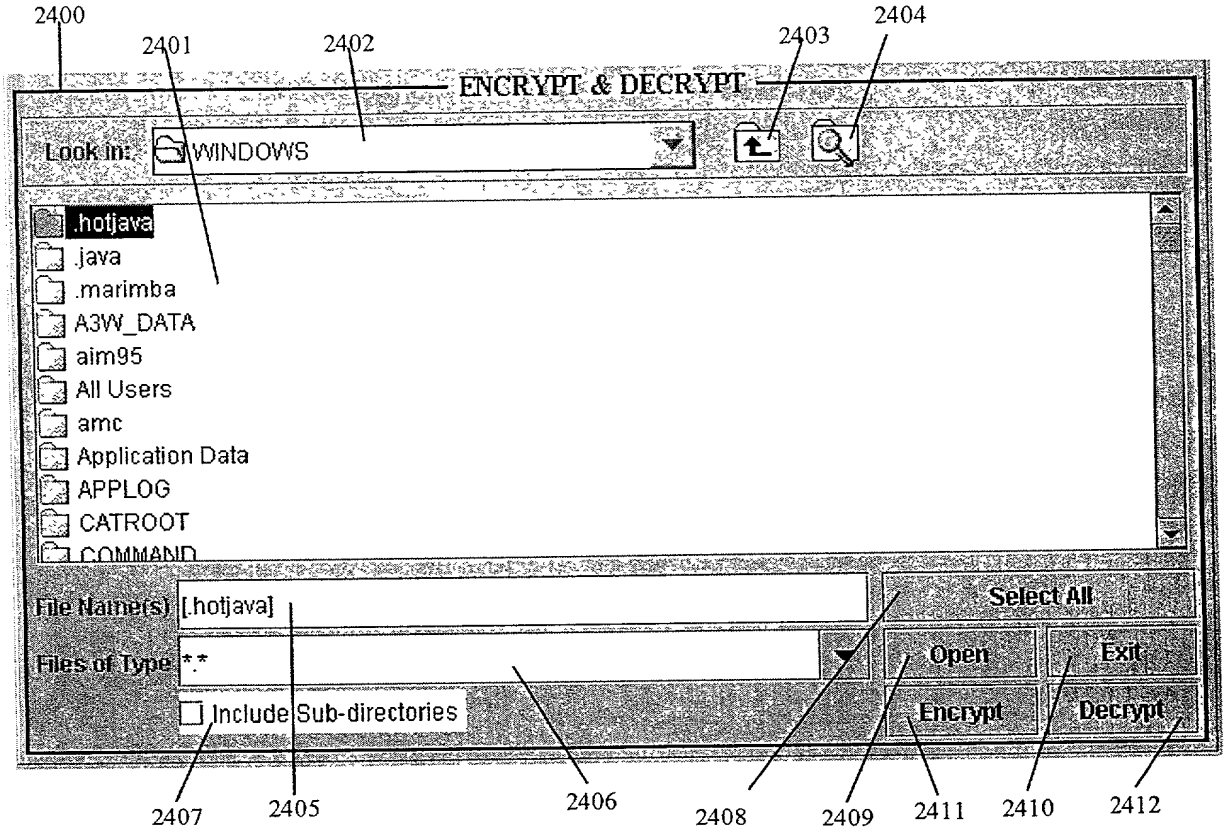

2400
2401
2402
2403
2404
ENCRYPT & DECRYPT
Look In:
WINDOWS
.hotjava
.java
.marimba
A3W_DATA
aim95
All Users
amc
Application Data
APPLOG
CATROOT
COMMAND
File Name(s)
[.hotjava]
Files of Type
.
Include Sub-directories
Select All
Open
Exit
Encrypt
Decrypt
2407
2405
2406
2408
2409
2411
2410
2412

Figure 34

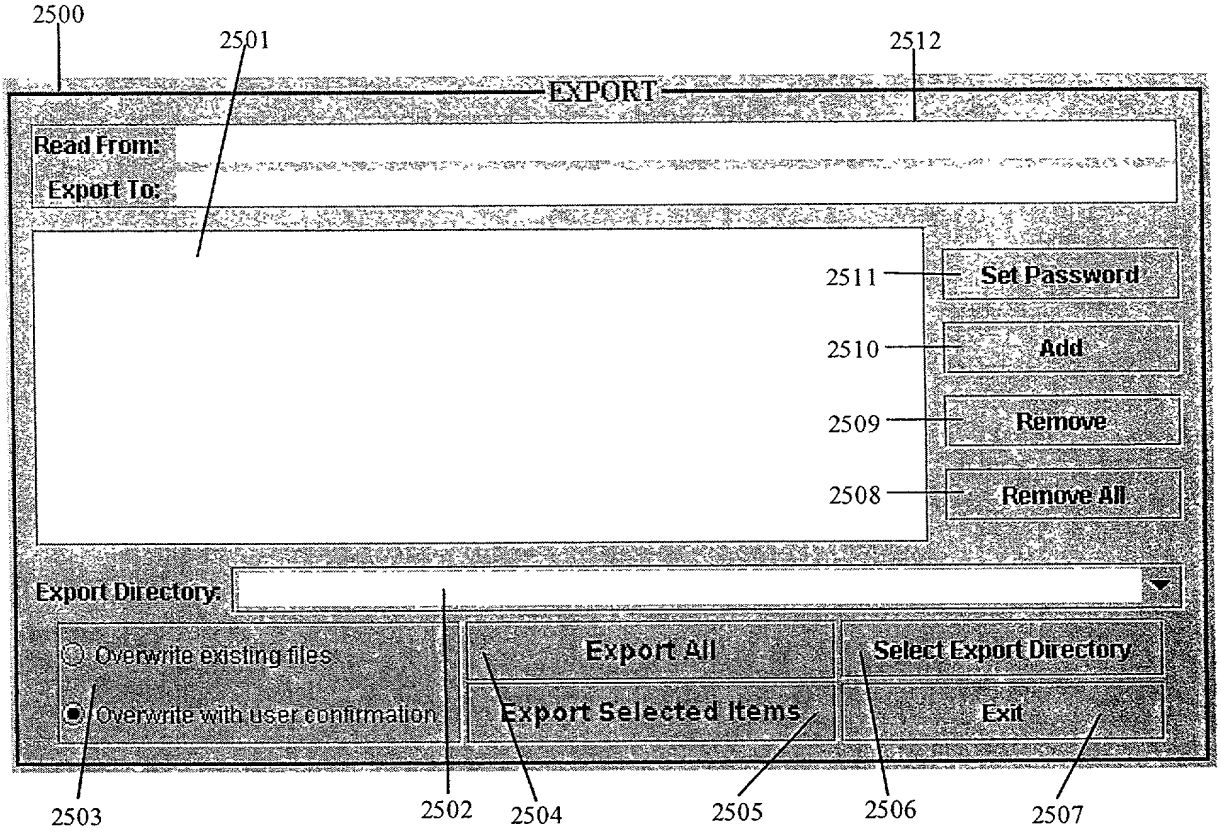

2500
2501
2512
EXPORT
Read From:
Export To:
2511
Set Password
2510
Add
2509
Remove
2508
Remove All
Export Directory:
Overwrite existing files
Overwrite with user confirmation
Export All
Export Selected Items
Select Export Directory
Exit
2503
2502
2504
2505
2506
2507

Figure 35

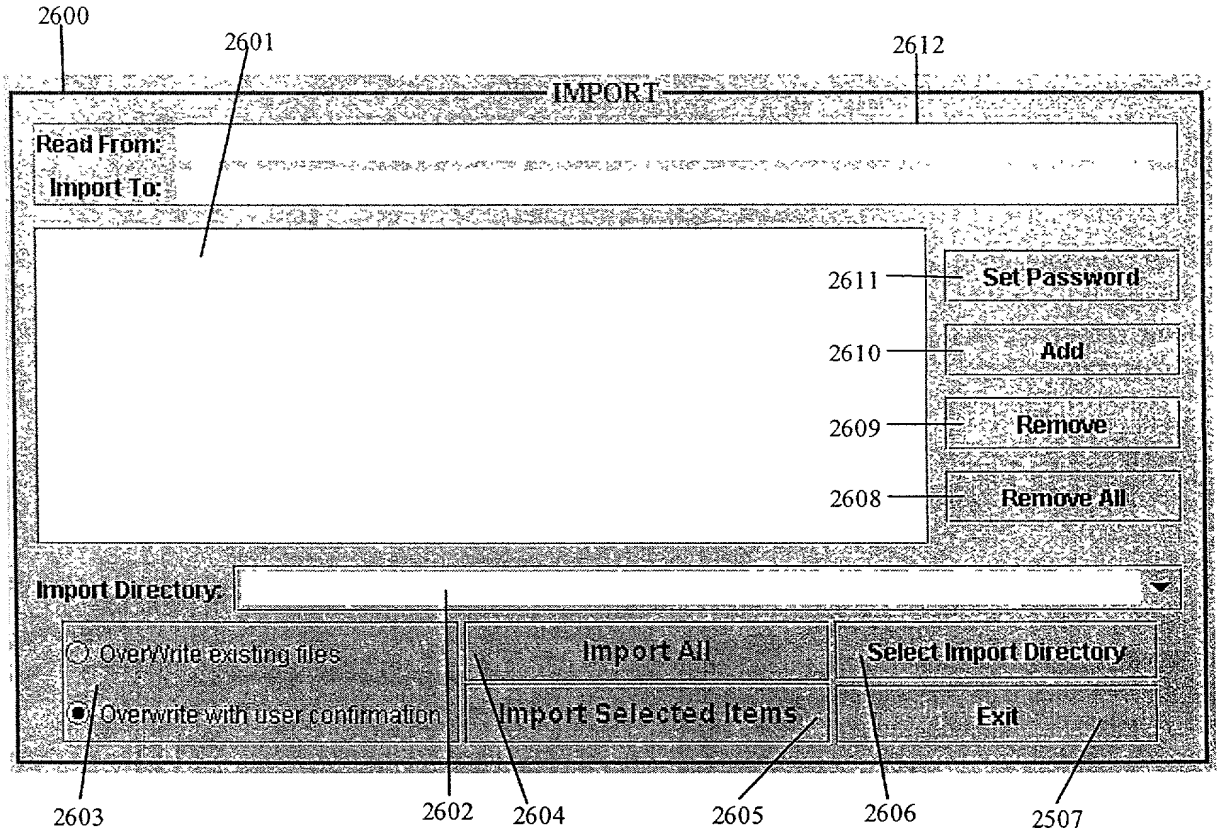

2600
2601
2612
IMPORT
Read From:
Import To:
2611
Set Password
2610
Add
2609
Remove
2608
Remove All
Import Directory:
Overwrite existing files
Overwrite with user confirmation
Import All
Import Selected Items
Select Import Directory
Exit
2603
2602
2604
2605
2606
2507

Figure 36

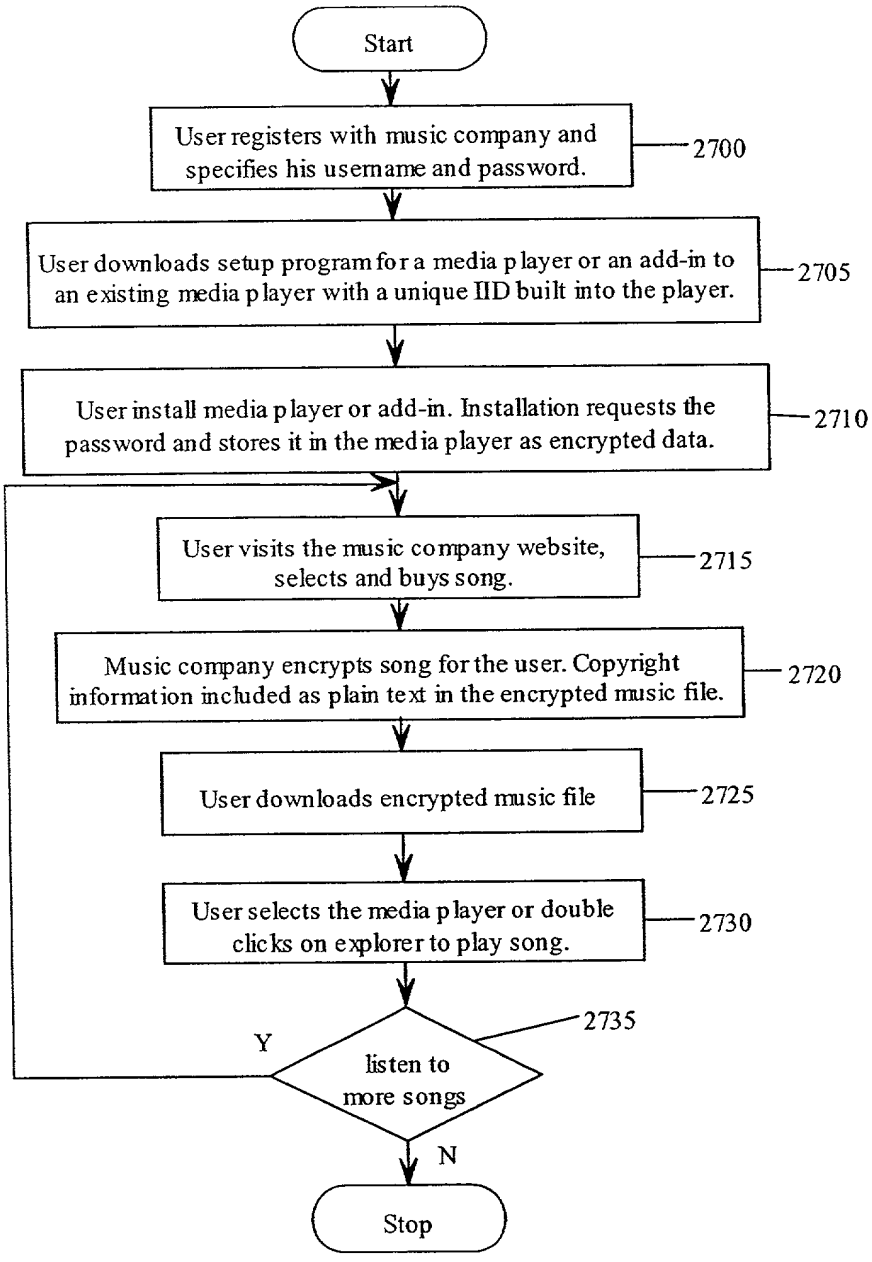

Start
User registers with music company and specifies his username and password.
2700
User downloads setup program for a media player or an add-in to an existing media player with a unique IID built into the player.
2705
User install media player or add-in. Installation requests the password and stores it in the media player as encrypted data.
2710
User visits the music company website, selects and buys song.
2715
Music company encrypts song for the user. Copyright information included as plain text in the encrypted music file.
2720
User downloads encrypted music file
2725
User selects the media player or double clicks on explorer to play song.
2730
listen to more songs
2735
Y
N
Stop

Figure 37